HW/ SW Performance Analysis using Virtual Prototyping Technologies

Mark Carey
Technical Marketing
Embedded Software Division

September 2014
HW/ SW Development Evolution

Hardware Software interaction presents a significant system validation challenge

Device

System / SOC
+ CPU(s)
+ Coprocessors
+ Graphics
+ Audio
+ Network(s)
+ GPIOs
+ Custom IP

Linux / RTOS
+ Boot code
+ Security layers
+ System config
+ Drivers
+ ISRs

Application Code

HW & SW Junction

Pain Here!
Embedded System Traditional Design Flow

- Late availability
- Limited visibility & trace capacities
- Complex setup & control
- Intrusive – Measurement affects behavior….
Physical Hardware Debug Affects Execution

A single “printf” statement adds 80us and increases runtime by 60%.

<table>
<thead>
<tr>
<th>Program</th>
<th>No Instrumentation</th>
<th>With Instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dhrystone</td>
<td>56,481</td>
<td>85,970</td>
</tr>
<tr>
<td>Stream</td>
<td>79,683</td>
<td>108,526</td>
</tr>
</tbody>
</table>

Kernel trace using LTTng (Dhrystone) slows execution
  - 3x performance impact
  - 52% greater image size

CoreSight enforces Design insertion, impacts power and is limited to designed parameters.

Debug services available only after boot loader completes.
Embedded System Virtual Design Flow

Changing the traditional design paradigm on prototyping boards

1. Model Creation
   - TLM Platform

2. HW Debug & Analysis
   - HW Debug
   - Performance
   - Power

3. ESL to RTL Verification
   - UVM
   - Scoreboard
   - Seqs
   - Agent
   - RTL
   - Agent

4. Virtual Prototype Creation
   - Virtual Prototype
   - SW Development

5. Software Debug
   - SW Debug

6. Software Analysis
   - HW/SW Analysis
   - Non-Intrusive Trace
TLM SoC Prototyping

- **Scalable TLM Modeling**
  - Standard TLM-2.0
  - Timing/Power attributes
  - Protocol Aware
  - Deterministic SW Execution

- **Native TLM-2.0 Debug**
  - Wave, Process Control

- **Advanced TLM-2.0 Analysis**
  - Throughput, Latencies, Cache

- **Virtual Prototype Creation**
  - Cross Platform

- **Linked w/ RTL/UVM Verification**
  - ESL to RTL Verification Flow
Building a Scalable Transaction Level Model

What is a scalable TLM?
- Separation of function, Timing/Power and communication layer
- Single scalable TLM supports LT (loosely timed) and AT (approximately timed)
- SystemC/TLM 2.0 OSCI compliant

Why Create a scalable TLM?
- Single scalable TLM can support all ESL use models

Algorithm Development
High Level Synthesis
Automated RTL creation
Usually C/C++ Function; Synthesizable for HLS

System Verification
Spec Validation
Exploration
Virtual Prototyping
Software validation
LT mostly sufficient for validation & comparative performance exploration

Architecture Design
Performance Optimization
Power Optimization
Software tuning
AT is required for accurate performance and power exploration & budgeting

Vista Scalable TLM
TLM Platform Validation and Debug

- Use embedded SW to verify the platform
  - Booted Linux, ran applications
  - Verified TLM platform behavior
  - Tested register/memory map

- Use real-world data driven tests
  - Interacted through terminal; displayed results

- Debug to “comprehend” the system
  - Look at design and code hierarchy
  - Track process/threads states during run
  - Switch between LT and AT during run
  - View transaction in LT and AT

TLM Socket Structure:
- TLM20 Generic Payload Fields: Address, Data, Size, Status, Commands...
- TLM20 Phases: Requests, Responses

TLM READ(Green)/WRITE(Red) Transactions Structure & Timing:
- GP Fields: Address, Data, Size, Status, Commands, TLM20 Phases Begin/End Request/Response

Switching on the fly between LT & AT for Debug & Analysis
Power Optimization Philosophy

- Use policies to model power of new IP
- Use accurate power models for legacy IP
- Analyze power profiles under
  - Typical system scenarios
  - Running software application
- Explore various power reduction techniques
  - Power domain management
  - Voltage/Frequency scaling

Platform Power Distribution
- Platform Complete Power consumption
- Power consumption of different instances

BUS Throughput
- The Complete Throughput
- Throughput broken to its specific contributors
Hybrid Co-Emulation Virtual Prototype

- **High Performance Solution**
  - Runs software at virtual Prototype speeds
  - Runs RTL at emulation speed
- **TLM platform capabilities maintained**
  - Performance and Power Analysis at the TLM
- **Matching configuration with simulation and emulation**
- **Hybrid Virtual Prototype configuration offering**
FPGA Bridge Interface

- FPGA Board runs the RTL sub-system of the Virtual Prototype
  - Validates new IP blocks in system context
  - Accelerates RTL execution speed

The combination of FPGA and simulation maximizes execution speeds and model availability.
ESL to RTL IP Verification

Step 1: UVM TB integration and debug with SystemC TLM

- SystemC TLM DUT reference model validated before being reused
  - TLM is now a TLM DUT

- UVM Block Level test bench constructed around TLM
  - Pseudo drivers send transactions to the model
  - Basic scoreboard functionality

- Proves
  - SystemC DUT model is correct with respect to the register description
  - Stimulus can reach acceptable functional coverage

- UVM Test bench can be developed, tested and debugged before RTL availability
ESL to RTL IP Verification

Step 2: RTL Block Level Verification with TLM Reuse

- TLM DUT is replaced with RTL implementation
  - SystemC DUT used as a reference model for the RTL

- Test bench and stimulus reuse from TLM test bench
Virtual Prototyping Attributes

- Virtual Prototype is a high-level simulation model of the target hardware
  - Can execute unmodified software images (drivers, firmware, OS, applications)
  - Runs at close to real time speed
  - Connects with standard tool chains and software IDEs
Virtual Prototype Creation

- Delivers a target HW model executable to the software team
  - Integrate final application software with actual hardware architecture
  - Validate and debug software against early HW model before RTL
  - Tune software to meet performance and power requirements
Instantly Configure Hardware

- You can easily manipulate and configure the hardware
- Test platform derivatives and run tradeoff analysis

Configure Cache, MMU, # of cores
Configure Memory & Controller
“Disable” selected IP’s
Run regressions on multiple alternatives
Manipulate clocks
Manipulate Connectivity
Tightly Controlled Execution

- When running a Virtual target the user gets full control over the hardware and software execution
  - Consistent hardware and software behavior
  - Well defined system Timing
  - Instantly stop all system clocks
  - Breakpoint on any hardware and software location
  - Freely step between hardware and software breakpoints
  - Runtime swap between functional & performance execution modes

Embedded SW Device

- UI™
- Application Stacks
- Middleware, Agents
- Android™ Linux® Others®

Interrupt Controller Timers

- CPU 0
  - L2 cache + SCU
- CPU 1
  - L2 cache + SCU

Peripherals

- USB
- ETHERNET
- SDRAM

© 2014 Mentor Graphics Corp. Company Confidential
www.mentor.com
Network & I/O Connectivity

“The Host” Machine

**SW Developer Tools**
- IDE & Builder
- Compiler
- Debugger
- Profiler

**Virtual “Target”**

- LCD Video
- MPEG
- GPU
- CPU
- Bridge
- DMA
- USB
- ETHERNET
- PCI EXPRESS
- SDRAM
- UART
- GPIO
- Timer
- PHY
- PHY
- PHY
- DDR3

**Peripherals**
- USB
- ETHERNET
- PCI EXPRESS
- Bridge
- Bridge
- USB
- Ethernet

**Terminal**

The Virtual Platform can connect through the host physical I/O’s
- Eth, USB, LCD, keyboard
- IP Address
- Ping from terminal…

**Embedded SW**
- UI™
- Application Stacks
- Middleware, Agents
- Android / Linux

**File I/O (gps, gyro)**

**Virtual Drivers**
Tightly Coupled Debug

View registers and variables in the hardware model

Break software execution on events in the hardware model

Virtual Prototype Manager
Non-intrusive Trace & Profiling

- Trace software without code instrumentation or compilation
- Unlimited real-time trace
- Manipulate software and hardware execution
- Inject hardware & software faults
- Set dynamically

Non-intrusive Trace

- Executed on Host
- TCL Scripts
- Code Injection
- Coverage Verification Profile
- Analyzer

- Registers & memories
- CPU Internals
- Caching
- Power
- Transactions

© 2014 Mentor Graphics Corp. Company Confidential
www.mentor.com
Combined SW & HW Analysis

- Allows to easily locate critical events in both regions
Multi-Core Analysis

- Access to all CPU’s system register layers, function scheduling, cashing and interrupt sequences

View function scheduling on each core

Functions running on each core

Call Graph on each core
Virtual Prototyping Key Values

✓ Availability before silicon or hardware are committed
✓ Instantly configure and manipulate hardware alternatives, derivatives and subsets
✓ Deep non-intrusive visibility to HW and SW
✓ “Unlimited” tracing capacities with no affect on behavior
✓ Tightly controlled hardware software execution
✓ Fast, deterministic execution of native embedded software
✓ Host resources provide real world I/O
✓ Its Virtual! You can share it over the network