Combining Pre-Silicon Verification Brains with Post-Silicon Platform Muscle

September 2010
Amir Nahir (nahir@il.ibm.com)
Agenda

- Background
- The Exerciser concept
- Using pre-silicon complex technique in the (simple) post-silicon environment
  - Through examples
- Focus on:
  - Functional aspects
  - Test generation
- Open question: the role of the acceleration platform
The Validation Effort: A Historical View

- **Methodology**
  - Test-plan
  - Functional coverage

- **Technology**
  - Genesys-Pro
  - Meteor (CSP, AI, ML)

- **Platform**
  - Simulation
  - Acceleration
  - Silicon

- Dedicated tests
- OS & Utilities
- Ad-hoc tools
- Exercisers
The Validation Effort: A Vision for the Future?!?!

Methodology

Technology

Platform

Simulation

Acceleration

Silicon

One technology

One methodology

A Bad Idea!!
## Platform Tradeoffs

<table>
<thead>
<tr>
<th></th>
<th>Software Simulation</th>
<th>Hardware Acceleration</th>
<th>Bring-up Silicon</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>slow</td>
<td>faster</td>
<td>ideal speed</td>
</tr>
<tr>
<td><strong>Control &amp; Visibility</strong></td>
<td>total model control/visibility</td>
<td>control/visibility with penalty</td>
<td>very limited</td>
</tr>
<tr>
<td><strong>Price per platform unit</strong></td>
<td>relatively inexpensive</td>
<td>expensive</td>
<td>expensive as a functional verification platform</td>
</tr>
</tbody>
</table>
Where is the Mismatch?

- Suggestion: let’s use pre-silicon test-cases, and run them on silicon

- Example - 10000 instructions long test-case:
  - Takes ~1min to generate
  - Takes ~1min to setup the silicon and load it
  - Takes <1ms to actually run it

- Machine utilization < 0.002%

- If setup time is dropped, generation becomes the bottleneck
  - Need > 60,000 servers to “feed” a single bring-up machine
The Validation Effort: A Vision for the Future!!

One methodology

Pre-Silicon Technologies

Post-Silicon Technologies

Simulation

Acceleration

Silicon
The Challenge

Simulation → Few cycles → Make the most of every cycle → Complex technologies

Silicon → Lots of cycles → Strong platform → How can we make the most of both?
The (Bare-Metal) Hardware Exerciser Concept

- A program that runs on the hardware and tests it

- Fully contained: does not interact with the environment
  - To reduce overheads
  - Load once, run “forever”

- Simple:
  - So it can run on early bring-up silicon
  - So we can debug it
  - To ensure high throughput

- Bare-Metal: no Operating System
  - To reduce overheads
  - To enable complete machine control
Threadmill: A Bare-Metal, Directable, MP/MT Exerciser

- Takes similar inputs to Genesys-Pro
- On-platform static generation
- Multi-pass checking
  - Has implications on generator component
- Distributed
  - Scalable

Exerciser Image

Builder

Generator & Kernel

System Configuration

Test Template

Architectural Model

Test Template

Architectural Model Testing Knowledge

Generation

Execution

Checking

OS services
Threadmill: Execution Stages

Off-offline stage

Build stage

Run-time stage

Utils

Data

Builder

System Configuration

Generator & Kernel

Test Template

Architectural Model Testing Knowledge

Architectural Model

Test Template

Topology

Architectural Model

OS services

Generation

Execution

Checking
Threadmill Stages: Affecting Factors

**Off-offline stage**
- No access to:
  - Config
  - Test-template
  - Params

  Can be as complex as possible

**Build stage**
- Reproducible
  - Reasonably Fast
  - Support randomness

**Run-time stage**
- Simple
  - Easy to debug
  - Reproducible
  - Fast
  - Small

Leeway for some complexity
Example I: Floating-Point Data

- Generating Floating-Point data operands is hard

- Pre-silicon tools make use of CSP, leveraging multiple techniques (analytical, stochastic, etc.)
  - Cannot be done (online) in a post-silicon tool

- Using basic randomization techniques simply isn’t enough

- Solution: import data operands generated (off-line) by a pre-silicon tool
Importing Floating-Point Data Operands

Diagram:
- **Builder**
  - **System Configuration**
  - **Generator & Kernel**
  - **FP Data**
  - **FP Data Table**
  - **FPgen**
  - **Test Template**
  - **Architectural Model Testing Knowledge**
  - **OS services**
  - **Generation**
  - **Execution**
  - **Checking**

Exerciser Image:
- **Test Template**
- **Topology**
- **Architectural Model**
- **FP Data**
Example II: Address Translation

- More complex than FP:
  - Control related, state dependant (e.g., 32/64-bit modes)
  - Shared between threads

- Pre-silicon techniques:
  - Rely on a reference model to track state
  - Instruction-by-instruction generation

- Solution: import data generated (off-line) by a pre-silicon tool
  - Threads rely on the same data
  - Cover all cases (multiple translations per real address)
Importing Address Translation Data

Diagram:

- **DeepTrans**
  - Translation data table
  - FP data table

- **FPgen**
  - FPgen

- **System Configuration**
  - Test Template

- **Generators & Kernel**
  - Architectural Model
  - FP Data
  - Translation Data

- **Builder**
  - Architectural Model Testing Knowledge
  - OS services

- **Exerciser Image**
  - Test Template
  - Topology
  - Architectural Model
  - FP Data
  - Translation Data

- **Generation**
  - Execution
  - Checking

- **FP Data**
Example III: Memory Management Scheme

- The underlying scheme which supports the load/store mechanism
- Basic requirements:
  - Random
  - Simple
  - Support for a large variety of events (collisions, crossing, etc.)
- To make things interesting:
  - Distributed
  - Reproducible
  - Configuration-dependant, test-template dependant
Example III: Memory Management Scheme (cont.)

- Relaxation: false-sharing only
  - To support simple checking
  - To avoid synchronizing the threads

- Pre-silicon techniques (logging, IBI generation) are simply too different

- Solution concepts:
  - Determine address ownerships offline
  - Enough memory, but not too much
  - Testing knowledge: to support a variety of events
Building Memory Management Data

- Each thread gets 50 intervals, such that no two intervals intersect
- Testing knowledge:
  - Cache-line crossing, Page crossing
  - Collisions
- And let the user define its priorities…

Solution: use CSP, with hard constraints to ensure validity and soft constraints to add the testing knowledge
Memory Management: Some Challenges

- Supporting online simplicity: through lookup tables

- Supporting online randomness:
  - Big table: probability of collision drops
  - Small table: quickly exhaust the address pool
  - Solution: Multiple tables of medium size with synchronized transition between tables
Importing Address Translation Data

DeepTrans

Translation data table

FP data table

FPgen

FP Data

Translation Data

Memory mgmt data

Builder

CSP

System Configuration

Generator & Kernel

Exerciser Image

Test Template

Topology

Architectural Model

FPgen

DeepTrans

Translation data table

FP data table

FPgen

Test Template

Architectural Model Testing Knowledge

OS services

Generation

Execution

Checking
Summary: Leveraging the Offline Stages

**Off-offline stage**
- Cover all cases

**Build stage**
- Filter
- Bias based on test-template
- Guarantee correctness of allocations
- Arrange data in tables and lookup tables

**Run-time stage**
- Basic random choices
- Lookups
- Simple math

**Examples:**
- For every mode, and every RA, at least two EAs
- FP coverage models
- Guarantee every thread has 5 addresses that support page-crossing
- Directive activation
- Calculate alignment
**Conclusion**

- Platform differences dictate the need for different technologies
- Post-silicon technologies must be simple
- Can still leverage complex pre-silicon technologies, by using them in the offline stages of an exerciser
An Open Challenge: the Role of the Acceleration Platform

- Growing importance:
  - Prepare tools for Bring-up
  - Enable software development
  - Support the pre-silicon bug-hunt

- The intermediary platform: fast simulation? or slow silicon?
  - Supports coverage and observability, but with penalty

- A platform on its own?
  - Can add (not too much) logic…

- Many open research questions!
Thank you
Backup Slides
Cross-Platform Functional Verification Methodology

Functional Verification Plan

Pre-Silicon Test Templates
- Genesys-Pro
  - Coverage analysis
  - Verification on Simulation

Post-Silicon Test Templates
- Threadmill
  - Coverage analysis
  - Verification on Accelerator
  - Post-Silicon Verification

Adaptation