Quality Improvement Methods for System-level Stimuli Generation

Roy Emek (emek@il.ibm.com)
Simulation-based Verification Technologies
Simulation-based functional verification

- Verification: Show that a design (implementation) conforms to its specification
- The main method today: Simulation

**Stimuli Generator** ➔ **Stimuli (test-case)** ➔ **Specification**

- Expected behavior
- Implementation
- Actual behavior

© 2004 IBM Corporation
Overview

- Systems and system verification
- The concept of Testing Knowledge
- Testing Knowledge mechanisms: three examples
- Testing Knowledge as a component in a verification methodology
- Experience
  - Functional coverage
  - A sample bug
- Implementation: constraints in Constraint Satisfaction
- Problems

- I too added a few slides …
A system:
- A configuration of various components
  - E.g., processors, memories, bridges, encoders, interconnect, …
  - Capable of interacting with each other and with the outside world
    - E.g., DMA of 1K bytes, decoding three MPEG frames, …

System verification: Verifying the integration of pre-verified components

Challenges
- Large designs
- Complex specifications
- Limited resources, specifically tight schedules
- Remoteness (physical, in time) from the actual logic designers

“Verification consumes ~70% of the design effort”
A major solution direction: reuse

TK represents another form of reuse
The concept of system-level testing knowledge

- Testing knowledge (TK): A set of mechanisms that aim at improving test-case quality
- Capitalize on recurring architectural concepts, such as:
  - Caches
  - Address translation and translation tables
  - Multiple instances of the same component type
- The basic mechanism: non-uniform random choice
  - Bias towards ‘interesting’ areas
- Affects all generated test cases
  - But can be controlled by the users of the stimuli generator

Space of valid tests

‘interesting’ areas
X-Gen

- **X-Gen**: A system-level test-case generator
  - An in-house tool
  - Used for the verification of several high-end systems in IBM

- The ideas presented here were developed during our work on X-Gen
  - Influenced by ideas from the processor verification domain
  - I.e. Genesys
Overview

- Systems and system verification
- The concept of Testing Knowledge
- **Testing Knowledge mechanisms: three examples**
- Testing Knowledge as a component in a verification methodology
- Experience
  - Functional coverage
  - A sample bug
- Implementation note: Constraint Satisfaction Problems
Testing knowledge Example #1: Resource contention

- System level ➔ multiple components ➔ parallelism
- Resource contention is a frequent cause of system-level bugs
  - Example: cache coherency and consistency
- Resource collision TK mechanism
  - Maintain a queue of recently accessed resources
  - With probability X, use one of the resources in the queue
- System-address is a typical system-level resource identifier
  - Resource contention ≈ address collision

![Diagram showing up 0, up 1, and DMA with resource contention]

© 2004 IBM Corporation
Testing knowledge Example #2: Address translation

- Address translation is a means for decoupling
  - Processors: virtual vs. physical
  - System address space vs. I/O address space
  - High-end interconnect
- ‘Placement’ events: relate to pages, segments, etc.

Found a complex bug in a high-end design
Testing knowledge as a form of reuse

- A test template describes a scenario / verification task
  - Example: 100 x transaction-A, then 50 x transaction B
- The same TK is reused across all the test templates
  - Thus reducing the number of test templates
- Similar testing knowledge can be used across multiple systems
A foreground / background methodology

Foreground: Main scenario
Defined by the test template

Example: a scenario that requires cache-misses would reduce the probability of address-collision

Bug

Background: testing knowledge
Intelligent random noise
Can be directed by the test-template
### Usage experience – Power4+ based system

<table>
<thead>
<tr>
<th>Category</th>
<th>TK based tool: X-Gen</th>
<th>Previous tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of request files*</td>
<td>737</td>
<td>7168</td>
</tr>
<tr>
<td>Simulation cycles (normalized)</td>
<td>x1</td>
<td>x4.8</td>
</tr>
<tr>
<td>Coverage Model #1</td>
<td>40.57%</td>
<td>37.10%</td>
</tr>
<tr>
<td>Coverage Model #2</td>
<td>43.84%</td>
<td>26.88%</td>
</tr>
<tr>
<td>Coverage Model #3</td>
<td>74.28%</td>
<td>63.80%</td>
</tr>
<tr>
<td>Coverage Model #4</td>
<td>61.14%</td>
<td>59.17%</td>
</tr>
</tbody>
</table>

* Rough measurement of the human effort
Implementation note: Constraint Satisfaction Problems (CSP)

- CSP definition
  - A set of variables
  - A domain of valid values for each variable
  - Constraints define valid combinations of values
- The basis for modern test-case generation
  - Constraints impose validity, user requests, aim towards quality
  - A test-case is a random solution
- Testing knowledge mechanisms are implemented as soft constraints
  - Reused for multiple systems
Summary

- Testing knowledge: Directing stimuli generation to ‘interesting’ areas
  - Expanding coverage
  - Increasing the chances of hitting a bug
- Capitalize on recurring architectural concepts
- Examples: resource contention, placement, interconnect

- Reduces the cost of implementing a verification plan
  - Reuse of knowledge between test templates
  - Reuse of knowledge (and technology) between different systems
- And at the same time influences the verification plan
Thank You