Deep Knowledge Test Generators & Functional Verification Methodology

IBM Verification Seminar
October 2003 - Laurent Fournier
All Starting Stages in Holding Position
Extreme Case of Influence on Sticky Bit

Rounder

1.00111011000111110010

I

1.00111011000111110001

1.00111011000111110001110000001110001000111

1.ccccccccccccccccccccccccccccccccc000000000000000000000001
Motivation for Deep Knowledge Test Generation

Gap + Bug-Prone

Control of test generator

Interesting scenarios
Deep Knowledge Test Generator?

Definition

- Test generation focused on specific verification areas, e.g. FPU, Microarchitecture Flow, SCU

Problem

- Inefficiency of generic architecture tools in specific error-prone verification areas (bugs not found or found late)

Objectives

- To provide greater control to reach non-covered areas
- To enable a systematic and comprehensive verification approach to speed up the rate of coverage
The Players

**FPgen**
- Generic, quasi-optimal solution for a well-defined, albeit complex, field
- Complex mathematic algorithms

**Piparazzi**
- An evolving solution to cope with microarchitecture flow complexity
- Classic constraint solving engine (CSP)

**DeepTrans**
- More a library of services for now
- Rounds off the model-based technology
Motivation: Floating Point Bugs

General bug curve

FP bug curve

number of bugs

time

number of bugs

time
Basic Functionality

<table>
<thead>
<tr>
<th>DIV</th>
<th>OP₁</th>
<th>OP₂</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[FP₁, FR₂]</td>
<td>0X1X001X10</td>
<td>0X0X001X10</td>
<td>00000000000 Max 4 bits set</td>
</tr>
</tbody>
</table>

Multiple solutions: Uniform distribution sought

Random solution: Uniform distribution sought
Piparazzi – The Main Concept

Solver

A test program for Micro-processor

iop[0].PIPELINE = FXU
iop[1].PIPELINE = LSU
iop[1].DISPATCH = iop[0].DISPATCH
lop[0].stage[STAGE_2].STALL > 0

Fetch

Decode

Dispatch
Input can be a Full Cross-Product

Example: All types model

Operand1

Multiply

Operand2

Result

+/- Infinity
+/- Zero
+/- Norm
+/- Denorm
+/- Large number
+/- Small number
+/- Min Denorm
+/- Max Denorm
+/- Min Norm
+/- Max Norm

+/- Infinity
+/- Zero
+/- Norm
+/- Denorm
+/- Large number
+/- Small number
+/- Min Denorm
+/- Max Denorm
+/- Min Norm
+/- Max Norm

+/- Infinity
+/- Zero
+/- Norm
+/- Denorm
+/- Large number
+/- Small number
+/- Min Denorm
+/- Max Denorm
+/- Min Norm
+/- Max Norm
Relationship between Generator Input Language and Test-plan

FPgen  ↔  Test-Plan

"Language shapes the way we think, and determines what we can think about" - B.L. Whorf
Generalized Piparazzi Example
Generalized FPgen Example

1.00111011000111110001110000001110001000111

Rounder

1.00111011000111110010

1.ccccccccccccccccccccc00000000000000000000

1.ccccccccccccccccccccc00000000000000000000

1.ccccccccccccccccccccc00000000000000000000

...
Generic Test Plans

✧ Methodology – Resource Dependent

✧ Density
✧ Crossing
✧ Reduction
✧ Huge models
Ingredients for Test-Plan

- IEEE test suites
- IEEE standard
- Bug analysis
- Papers
- Test plans from users
- Generic & alternative Implementations
The Evolution of Functional Verification Methodology

Small design: manual to reach all suspected cases

Slow and tedious
The Evolution of Functional Verification Methodology

**Increased complexity: Random generators**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="" /></td>
<td><img src="image2" alt="" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image3" alt="" /></td>
<td><img src="image4" alt="" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image5" alt="" /></td>
<td><img src="image6" alt="" /></td>
<td><img src="image7" alt="" /></td>
<td><img src="image8" alt="" /></td>
<td><img src="image9" alt="" /></td>
<td><img src="image10" alt="" /></td>
<td><img src="image11" alt="" /></td>
<td><img src="image12" alt="" /></td>
</tr>
</tbody>
</table>

**No uniformity in bug location probability**
The Evolution of Functional Verification Methodology

Biased Random generator to control towards suspected areas

Faster and more effective
The Evolution of Functional Verification Methodology

Deep Knowledge Test Generators

Many fast accesses to areas with a high probability of bugs
Coverage by Generation

Test Plan

Coverage by generation:

Def-Files

Writes Def-files directly asking for model

DKTGs
(FPGen, Piparazzi)

Coverage tools
(Meteor)

Writes specific Def-files to cover missing items
DKTG Performance

- Broad approach TG
  GPro, AVPGen
- Manual testing
- Control

Speed
Coverage Graph

- Generation method
  - DKTG
  - GPro

- % events tested
  - 100%

- Time
Cross-Product Approach

- Bugs often lie in the interaction of several factors.

- This approach is more than a list of disparate tasks. It may include, often inadvertently, many “quasi corner cases”.

- All types model.

- Some cases are clearly corner cases, but others, while interesting, might have been overlooked.
Quasi Corner Cases

- Straightforward cases
- Quasi corner cases
- Corner cases

Less interesting → More interesting
The Non-Uniform View
Pitfalls

✦ Quantity at the price of quality
  ✷ Easy to create large, not meaningful event spaces
    ✷ More events than can be covered (waste of verification bandwidth)
  ✷ Leads people to be “thought-lazy” because easy to generate impressive test-plan (quantity-wise)
  ✷ Blindly rely on “nice” coverage numbers
✦ Includes many events that require significant effort in knowing whether or not they are reachable (double-edged sword)
Conclusion: The Test-Plan Feedback Loop

Generic test plan → Updated tool development
Conclusion: DKTG  Impact

Test Plan Scope

Existing Verification Means

DKTG

Knowledge & Control
Technology Perspective

Design complexity and scope of test plan

1988
- Manual testing
- Slow, tedious, small design

1994
- RTPG
- Productivity
- Randomness

1998
- Genesys
- Core generation
- Model based
- Generic

2001
- Genesys-Pro
- Cross-Products
- Extended control: Randomness vs. specific
- Bug prone area
- Focused approach
- DKTG