
Using Java for VLSI Development

and/or

Advanced Open Source Java Software Engineering

Joseph R. Kiniry

KindSoftware, LLC and Fulcrum Microsystems, Inc.

Java for AVLSI?

- What is AVLSI?
 - Delay insensitive circuits
 - Power invariant
 - Design scalability
 - Process invariant
- How is VLSI design typically done?
 - High level specification (e.g., VHDL)
 - Low-level specification (e.g., Verilog)
 - Automated layout
- How are we doing it?
- How is Java used?

Challenges

- Performance
 - You try simulating a CPU in Java!
- Scalability
 - Massive memory and thread use
- Robustness
 - If simulation takes five days and it crashes on day four...
- **Correctness!**
 - Fabricating a chip = no patches

Design Process

- Multiple specification levels.
 - Multiple Java realizations
 - CSP (Concurrent Sequential Processes)
 - Production rules
 - Layout
- Unit testing
 - Cell, Unit, CPU
 - With and without OS
 - At multiple refinement levels
- Cosimulation for behavioral equivalence
 - Formal refinement checking

Observations

- (Mis)use of concurrency
 - Thread per anything
- Data structure (ab)use
 - Generic Java data structures
- Aimless optimization
 - (Aimless) optimization is the root of all evil.
- Untracked requirement changes
 - Complexity has a requires clause
- Documentation process
 - Least favorite part coupled with rapid corporate development and hard delivery dates.

Response

- Commercial tools where necessary
 - Analysis: JProbe and JProfiler
 - Revision control: P4 (was CVS)
 - Simulation: Cadence
- Open Source tools where possible
 - Custom code coverage: Gretel
 - Metrics: Java NCSS, SlocCount
 - Documentation: SGML and L^AT_EX@
 - Specification: JML
 - Build system: Ant
- Process, process, process
 - Documentation
 - **Specification**

Results

- Performance
 - Typical: 10 minute change = 10 percent
 - Atypical: one man month = 1000 percent
 - ...and nothing in between.
- Memory use
 - Garbage collection (ab)use
 - Iterators, Events, and StringBuffer
 - OS VM abuse
 - Overall memory size
- System monitoring
 - Subsystem tailored to design space
 - Optional compilation
 - Framework licensing (IDebug)

Key Aspects

- Lightweight specification
 - Semantic properties via Javadoc
 - Standard system overviews
 - Abstract
 - Overview
 - Requirements
 - Dictionary
 - System tracking
 - Development state
 - Deliverables
 - Tasks
 - Risk analysis
- High-level specification
 - Extended BON

Key Aspects, cont.

- Detailed specification
- Several alternatives, OS and commercial
 - iContract, Jass, JML, MetaMata, jContract
- Design by Contract as a start
- Model-based specification for completeness
- JML = Java Modeling Language

Semantic Properties

- Domain-specific specification constructs that augment an existing language with richer semantics.
- Used in system analysis, design, implementation, testing, and maintenance via documentation and source-code analysis and transformation tools.
- Three forms: *informal*, *semi-formal*, and *formal*.
- Advance over existing work because:
 - Specified as annotations, which is popular with programmers.
 - More natural model because little-to-no math.
 - Higher-level constructs, thus more expressive.

Example of Annotated Java

```
/**
 * Returns a boolean indicating whether any debugging facilities
 * are turned off for a particular thread.
 *
 * @concurrency GUARDED
 * @require (thread != null) Parameters must be valid.
 * @modify QUERY
 * @param thread the thread to check.
 * @return a boolean indicating whether any debugging facilities
 * are turned off for the specified thread.
 * @review kiniry Are the isOff() methods necessary at all?
 */

public synchronized boolean isOff(Thread thread)
{
    return (!isOn(thread));
}
```

Extended BON, cont.

- Extensions - semantically well-understood best practice programming constructs in specific domains.
 - *concurrency* - sequential, concurrent, guarded.
 - *temporal logic contracts* - for run-time testing.
 - *modifies* - predicates about state change.
 - *generates* - concurrency dynamism.
- Bijective refinement.
 - specification changes \leftrightarrow program changes.
- “Weak” language plus quality tools \Rightarrow stronger system than most of industry’s best practices.

Extended BON

- A lightweight specification language for software.
- A design model checker for multiple languages.
- Extended BON = BON + semantic properties.
- BON - the Business Object Notation.
 - Textual and graphical syntax.
 - Well-understood semantics.
 - Seamless, reversible specification language.
- BON specifications capture important system aspects...
 - static and dynamic structure, contracts, test cases.
- ...but are not complete and are relatively weak.

JML

- Syntax is extension of Java
- Contracts a la Eiffel
 - requires, ensures and interplay with inheritance
- Side-effects (or lack thereof)
 - pure, assignable
- Refinement (spec to impl, spec to spec)
 - also, refines
- Specification scoping (contract visibility)
- Model variables
 - Software cosimulation
- Documentation generation, static analysis, and dynamic run-time checks

JML Tools

- JMLDoc
 - JML plus Javadoc
- Typechecking
 - jml
- Compilation
 - jmlc
- Static analysis
 - ESCJava, LOOP, Daikon, etc.

For More Information

- JML

- <http://www.jmlspecs.org/>

- Extended BON

- <http://ebon.sf.net/>

- Fulcrum Microsystems

- <http://www.fulcrummicro.com/>

- KindSoftware

- <http://www.kindsoftware.com/>

- IDebug, semantic properties, code standards, formal methods and advanced software engineering consulting, etc.