



Test Benches: The Dark Side of IP Reuse

Gregg D. Lahti
Intel Corp



Reuse: The Holy Grail of Design Engineering

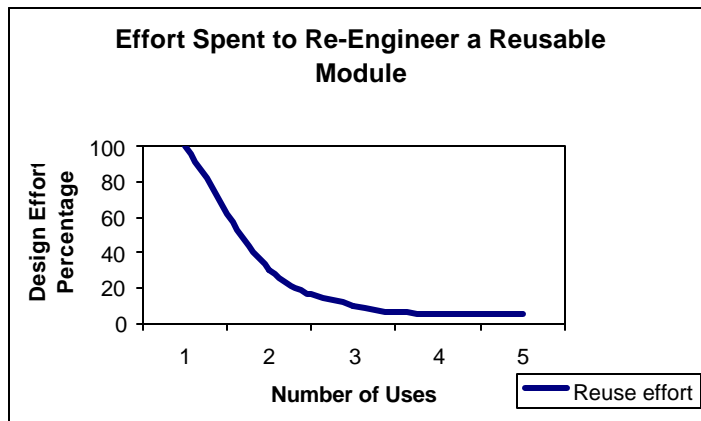
2

- What is design reuse?
 - Creating modular IP to be used in other designs
 - Utilizing IP from an existing design into a new design
- Design reuse buys you:
 - SoC (System On Chip) solutions utilizing common modules
 - Faster time to market for new designs
 - Validated IP (been through silicon & validation phase)
- Design reuse implementation costs:
 - Architecture costs (standardized vs. custom design)
 - Design time up front to implement reuse
 - Creates legacy



When Will Reuse Pay Off?

- Surprise! Reuse isn't free!
 - Reuse gains are not seen until after the second iteration
 - Future iterations of a design still require manpower for implementation, bug fixes, new target libraries and processes, etc.
 - Reuse must be initially architected into the design, test bench, tests





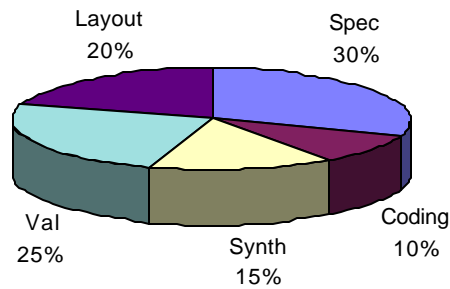
What Really is IP?

- Most consider the RTL description of the module
 - Patent lawyer and pointy-haired boss approach to IP
- In reality the IP of a design consists of:
 - RTL description of module
 - Documentation of the module
 - Synthesis constraints of the module
 - Test bench and tests for the module
- The tests are the most important aspect of the IP
 - More man-effort to write the test bench and tests
 - Validates the IP in various abstractions (behavioral, RTL, netlist)
 - Provide a compliancy test



Time Well Spent?

- More engineering time is spent validating than writing an RTL description and synthesis
- At least a 1:1 validation to design engineering task ratio, in some cases more



Breakdown of Engineering Effort



Why IP Reuse Fails

- This usually fails due to the “Designer’s Sandbox Rule”
 - Existing design needs modifications, so doing it from scratch would be easier: it is difficult understanding a design done by other engineers
 - Design, test bench, and tests may not be well documented
 - Existing design not setup for reuse- custom logic, instantiated cells, etc. Reuse needs to be architected up front, legacy blocks are not setup as reusable
 - Reusing a design isn’t glamorous: the chop-and-sim method of design isn’t fun or creative
 - Hint of doubt- are the existing design & tests good, can they be improved, any unknown bugs?
 - Test bench and tests are not setup for maintainability
 - Test bench and tests setup for system-only testing, fail to provide module-oriented tests



The Ties That Bind IP

- The RTL & tests get directed to a “point-solution”
 - Engineers design to a specific frequency/process, create IP with workarounds and legacy-binding items
 - Instantiated gates from a specific process
 - Tests written for use with specific frequencies, simulators, OS platforms
- Test benching isn't a fun job
 - Engineers generally don't like writing test benches and tests
 - Survey of IHDL '99 Conference by Cliff Cummings showed a small handful of engineers (less than 10%) like writing test benches



Reusable Test Benches

- Reusable IP requires the following:
 - A document describing the design at a micro-architecture level
 - The RTL description of the design with no instantiated gates and hopefully well-commented code.
 - The synthesis scripts, possibly oriented to a generic or included library, with documentation of all constraints, false paths, and clocking information.
 - **A self-checking, easily modifiable test bench that is well documented.**
- Just like RTL designs, test benches and tests need to be architected up front for reusability
 - Requires high-level description for tests
 - Requires modularity and portability



Poor Forms Test Benches

- Bad test bench methods which kill any attempts for reuse
- Vector stimulus
 - A set of 1's and 0's, applied for stimulus input and clock/frequency dependent output comparison
 - Lowest level of testing, worst for re-use
 - Not easy to debug or modify
 - Clock and frequency dependent- cycle slips due to changes in the design are hard to add or fix
- Assembly language driven tests
 - Test bench usually implies whole system (microprocessor, system controller, program memory) for testing environment
 - Test and test bench tied microprocessor family, not easily translated to another CPU/uC language

Poor Forms Test Benches (continued)

- Assembly language driven tests (continued)
 - Lower form of tests (large amount of test code per high level of operation)
 - Hard to create high-level functions with a low-level language (i.e. a memory write may take 5 assembly instructions)
 - Not as easy to debug
 - Ties the user to a specific platform for compiler (Solaris™ or HPUX™, or the worst, NT™)
- System-specific Test Bench
 - Similar to using assembly code driven test bench, where entire system must be represented in test bench for tests
 - Not very portable, large amount of overhead if using just a small part
 - Hard to manage, hard to integrate into new project testing environment

Poor Forms Test Benches (continued)

- **Scripts and Environments from Hell**
 - The use of many custom scripts to create the test bench and a poor environment to run them
 - Scripts are home-grown with little regard to reuse
 - Most of the time in perl, but could be in not-often-used language (Awk, python, vbscript, etc)
 - Hard-coded paths
 - No revision control on scripts
 - No common project area for scripts/binaries
 - Little or no documentation for environment, scripts
 - Environment is hard to “gather up” to reuse the test bench and tests for a module
 - Significant manpower to maintain, modify, or translate for module reuse

What Makes a Reusable Test Bench

- Test bench and tests in widely used, industry standard RTL language
 - Verilog, VHDL, C/C++
 - Eliminates need for expensive, custom test bench tools
- High level testing methods
 - Use a procedural-type coding effort with Bus Functional Models (BFMs)
 - More test operations for the test code effort
 - Easy to code, easy to modify, easy to debug
 - Very portable, uses native simulator language for test benching methodology

Verilog: SNUG'99 paper, paper A BFM Simulation Strategy for Verilog by Rodney Pesavento and Michael Privett

VHDL: SNUG'99 paper, Designing Procedural-Based Behavioral Bus Functional Models for High Performance Verification by Gregg Lahti and Tim L. Wilson



What Makes a Reusable Test Bench (continued)

13

- Using C/C++
 - Widely known and used in industry
 - Leverage work from SW development in SOC solutions
 - Good for test code if programming concurrent events not an issue
 - Supported via PLI 2.0 and OVI compliant Verilog simulators
 - VHDL simulators at an industry impasse:
 - No OVI-type standardization for C/C++ interface, causing portability issues between VHDL simulators (Modelsim™, Leapfrog™, VSS™)
 - Recommend using C/C++ with Verilog simulators only

What Makes a Reusable Test Bench (continued)

- Limit environment and scripts
 - Use widely-available, UNIX tools (Make, CVS, gcc, etc)
 - Tools are industry proven, generally supported by the industry
 - Tools are generally cheap (GNU, Linux, etc) and easy to get
 - Be smart about script creation
 - Avoid hardcoded paths, use environment variables
 - Use setup script for user environment, write it with portability in mind
 - Get rid of patch scripts once EDA tools are fixed
- Keep it stupid simple
 - Complex test benches require manpower overhead to maintain
 - Complexity is hard to learn, hinders future reuse of modules
 - Goal of test bench and tests is to prove functionality and provide a fault grade of design



Why Not Vera™?

- Vera™ is a good concept which forces high-level testing methods on the user
 - Procedure based tests
 - Higher abstraction level of test code
 - Useful mailbox, branching allocation, and randomized data features
- Vera™ fails in portability
 - Requires a license for use
 - Language is not widely used, open to industry
- Vera™ is a new language
 - “Newness” issues of syntax, annoyances, bugs, and workarounds in programming language
 - Language not widely accepted, not many engineers proficient in Vera
 - Yet another language to learn

Why Not Vera™? (continued)

- Does the industry need another language?
 - Can you do the same functionality in Vera™ that you can do in VHDL, Verilog, C/C++?
 - 11/01/98 EE Times article Babel of languages competing for role in SoC by Richard Goering and Peter Clarke, wrote this quote:

"There are a number of competing approaches to system-level design, but the only realistic candidate in the context of SoC design is C++," said Synopsys' Brian Barrera. "Proprietary languages do not easily connect with C/C++, the de facto standard languages for systems and software development. Proprietary extensions to C are not appropriate, because they lock you into a single supplier."



Conclusions

- Reuse can be a key element in achieving SoC designs with a fast TTM
- Over half of the logic design effort is in the testing and validation of the RTL
 - Large reuse gains in terms of productivity
 - Great for reuse guidelines to be followed
- Key items for a reusable test bench
 - Tests and test bench in industry standard, widely known RTL language
 - High-level abstraction by use of procedures, functions, and tasks
 - “Open” environment: limit scripts, keep to UNIX-type, industry-standard languages and tools
 - Keep it stupid simple