How to Start Verifying A Processor

Paul Tobin
Sr. Manager, Verification
AMD
Paul.tobin@amd.com
Send In The Advance Team

Engage seasoned verification experts during early design
• Peek in when performance modeling is in full swing
• Iterate informally with the designers, getting an idea of what’s to come
• Be on the lookout for potential bug farms

Close this effort with architectural documentation
• Review the proposed design broadly, including verification staff still on the previous project(s)
• RTL does NOT count as documentation ("I already wrote the code, why should I document it?") !!

Identify the big ticket verification issues
• What significant new checkers or entire testbenches are required? Is the feature they’re verifying worth the cost?
• Are there new tool releases, or entirely new tools, that have to be qualified?
Answer the Perennial Questions

Should we use a HW emulator/accelerator?
- Only if HW/SW co-verification is critical, and you use the box from day 1
- Not useful simply for “more simulation bandwidth” or “really long tests”

Do we verify performance?
- Pick your battles: invest in highly specific performance checkers, or high level correlation.
- Don’t replicate the RTL with timing accurate C++
- Correlate trace “snippets” to a high level performance model, if available

Can we verify power?
- Some 3rd party tools are available, but quality power estimation from RTL is an elusive problem.
- This is akin to performance verification, where highly specific checkers can be valuable.
Push Methodology That Impacts RTL

Close quickly on things that dictate how RTL code is written

- **Assertions:** the obvious one today!
  - Agree on which syntax to use
  - Which assertions RTL inserts vs. what verification will add
  - Who’ll debug assertion firings?

- **Front end linters:** tools that check RTL syntax minimally, and semantics if possible

- **Design modularity** – how will RTL model:
  - Flops and latches. Instantiated or inferred? What does DFT want? What does the equivalence tool want to see?
  - Memories
  - Asynchronous clock domain crossings and synchronizers
  - How many modules per file, how many levels of hierarchy?
  - Create global include files for paths to signals that verification will monitor
Is It Too Late For Methodology Change?

Verification is usually busy taping out the last chip while RTL is designing the next one. By the time we get there, the window for “big investigations” has closed...

Realize that methodology advancement must be continuous, not an impulse function. How do we avoid getting stuck in ruts and firefighting through project after project?

- Always have some team members looking forward
- Review results & proposals 3-4 times a year
- Take risks! Don’t wait for absolute guarantees of success...
What Are The Untapped Opportunities?

Fodder for “R&D” evals these days include:

- SystemC for testbench support, including parallelism like HW languages, and the full power of C++. No environment is a panacea, but rolling your own has hidden costs...
- System Verilog for the testbench. Do you trust all of the features yet? Will it really improve productivity, or is it just “cool”? 
- New breed of fast cycle based simulators. They didn’t get much traction in the 90’s, but are back in vogue again.
- Static analysis techniques, especially leveraging assertions. Find engineers with the right mindset and talent, as these are not general purpose tools and will fail in the wrong hands.
- Automatic assertion generation