Jump-Start Software-Driven Hardware Verification with a Verification Framework

Matthew Ballance
matt_ballance@mentor.com

Hardware Verification

- Verify functionality from outside the design
- Lots of parallelism
- Testbench wraps around the IP
- Verification frameworks like UVM boost productivity

Software-Driven Hardware Verification

- Verify integration of IPs from inside the design
- Lots of parallelism
- Test software typically “bare metal”
- Labor-intensive to create test variants

Software-Driven Verification Challenges

- Lacking OS-provided services
- Reuse is ad-hoc
- Little support for test-creation automation
- Little support for easily creating test variants
- Minimal infrastructure
- Resource-constrained environments

A Framework for Software-Driven Hardware Verification

- Facilitates encapsulation and reuse
- Provides key infrastructure
- Enables automation

Component Model and Factory

- Interfaces
- Enables test functionality to be encapsulated
- Factory makes it easy to create test variants
- Build, connect, execute phasing

Delegated Implementation

- UVM delegates implementation via transactions
- Software delegates implementation via APIs
- SVF provides uni- and bi-directional API delegation

Efficient Logging

- “Test” code clearly separated from “driver” code
- Efficient logging
- Global checking via analysis API ports
- Easy to create test variants by replacing components

SVF provides uni- and bi-directional API delegation

Efficient Logging

- Message Format
- Message Data
- Logging is critical for analysis and debug
- Logging introduces significant overhead
- SVF splits message format, arguments for efficiency
- Logging and message rendering separated for efficiency

Conclusion

- Verification frameworks boost productivity
- A software-driven verification framework
  - Facilitates component encapsulation and reuse
  - Tailored to specific needs of software-driven verification
  - Enables automation
  - Simplifies creation of test variants