SoC HW-SW Co-Verification Challenges

Sudhakar Surendran
Technical Lead
MCU, Texas Instruments
Agenda

• SoC HW – SW Co-verification

• Challenges in:
  – Verification plan creation
  – Testcase creation
  – Testcase execution
  – Debug
  – Coverage closure

• Way out!

• Conclusion
SoC HW – SW Co-verification

• 71% of SoCs contain at least one embedded processor*
  – In advanced SoC it can be as many as 20

• Pre-Si Verification: Not just for HW but for SW too!
  – Ensure SW is ready when Si is ready to sample
  – Dependencies due to complex clocking and power management
  – Identify bugs due to SW use scenarios
  – SW is burned in ROM

• HW-SW Layers


18-June-15
Verification Plan Creation

- Scenarios for Register and Functional layers easy to identify

- Scenarios identification for Peripheral sequence and Application usecase is challenging
  - No clear specification
  - Multiple combinations exists for the sequence and usecase

- Use divide and conquer approach
  - Identify scenarios as per functionality e.g., power usecases, video usecases, imaging usecase, etc.

- Get more clarity on usecase
  - Previous generation SoC end equipment usecase analysis
  - Use knowledge of marketing, system architect, early customer interactions
  - Reverse engg. existing SW code e.g., Windows driver knowledge to get usecase for PCI peripheral used in PC/Laptop end equipment

- Avoid duplication between functional verif. and HW-SW verif.
Testcase Creation

• Library creation/verification
  – Defined, developed, verified and used simultaneously
  – Clear process on when, how and what libraries to be used
  – Tied closely with testcase automation/randomization

• Challenges in automation and randomization
  – Random : To be or not to be !!
    • Randomize control and/or data ?
    • Testbench enabled for randomization ?
    • Usually random at peripheral sequence layer and a mix of random and directed at usecase layer
  – Choose tool based on veri. plan : Homegrown, Graph based, Perspec™, …
    • Scheduler for homegrown setup
  – Testbench updates for HW-SW co-verification
  – Define APIs and their interfaces upfront to enable co-development
  – Developed on non-mature design : Testcase bug or design bug ?
  – Reusable post silicon?
Testcase Execution

• Run time issue: Large data and complex usecase

• Choose right platform: Acceleration, Emulation, FPGA prototyping
  – Special skill to setup the design on the chosen platform
  – Design preparation for the chosen platform
    • Changed or Modeled: Clocking, memory, debug interface, analog
    • Verifying the changes and maintaining the accuracy?

• Realtime or synthesized data?
  – Realtime may need special interfacing to meet speed requirements e.g. Palladium speed bridge
  – Repeatability of the data sequence to enable debug

• Custom board to interface with the platform chosen
  – Has required interfaces and components to verify the planned usecase
  – Capability to support randomization
  – Reusable post-silicon
Debug

• SW debug challenges
  – Bottom up verification approach helps. But, co-development does not allow this
  – Problem of huge & plenty – huge number of random tests debug!
  – Adding debug statement can alter timing and state and miss the bug!
  – Use advanced debugger features – trace, conditional breakpoints / watchpoint, etc.

• HW debug challenges
  – Control and Debug ability decreases as you go up the platform chain
    • Simulation → Acceleration → Emulation → Prototype → Silicon
  – Control and repeatability of data stream is important!
  – Restarting from the initial state can be a challenge
  – Co-development leaves stale design for HW-SW verification
    • Bug already found in Verification!
Coverage closure

• Functional coverage is important metric for HW-SW verification
  – Structural coverage does not add value

• Need abstract functional coverage bins
  – Conventional functional coverage bins using registers/sequence does not make sense
  – E.g. H.264 video X 1080{p,i} X {30, 60, 120, 240} Frames X PowerDown
  – Conventional functional coverage not be possible in higher platforms e.g. FPGA

• Reporting and summarizing abstract functional coverage
  – Use embedded print statement in SW to report coverage
  – Summarizing done using custom scripts by parsing logs
  – Some random test generation tools support coverage reporting & collection

• Not just for random but also for directed tests to measure what has been accomplished
Way out!

• Planning, Planning, Planning
  – Necessitated by co-development and compressed schedules
  – Significant upfront planning and co-ordination with all stake holders
    • Spec, System, Verification, SW, ROM, Validation, Design teams
  – Resource planning
    • Debugger, Emulation/Prototype resource, human resource

• Executing on smaller pieces that form the big picture
  – While waiting for design to be matured
  – Verifying the SW libraries
  – Verifying the design changes for higher platforms
  – Verifying toy applications

• Multi-skilled team
  – Verification, SW, Application, prototype
Conclusion

- Pre-Si Verification is for HW and SW
- Numerous challenges exists in every step
  - Solutions exits too
- Proper planning and execution helps us to get there
- Dedicated multi-skilled team can get us there
- Customer success depends on the how good HW-SW verification was!
- HW-SW co-verification is a well understood problem
- Emerging challenge: SoC Analog co-verification