CONNECTING THE DOTS: APPLICATION OF FORMAL VERIFICATION FOR SOC CONNECTIVITY

Bin Ju, Staff Application Engineer, Cadence Design Systems, Inc.

Introduction
- Real design case study with unnamed customer in 2013
  - Customer personally supported by the author
- Customer experiencing greatly increased complexity in each SoC generation
- SoC top-level design integrates increasing number (instances and types) of IP blocks
  - Third-party and in-house IP
  - Some connected directly to protocol-based on-chip bus
  - Others connected via glue logic
- Always glue logic between IP interfaces and IO pads

1a: Configuring Property Generation
- Assertion Language
  - SV/PSL/TLV
- Language at top-level design
  - Verilog/VHDL
- Property File Name
  - Define for the Property File Name
- Default Clock
  - Sampling clock for properties
- Default Delay
  - Reset Condition used for disable/enable
- Property File
  - Delay value for pipelined connections
  - Needed for pipelined connections only
- Toggle Checks
  - Enable/Disable Toggle Cover generation
- Toggle Checks
  - Sampling clock for Toggle covers

1b: Setup Table – Specifying Pads
- Used to define pad types and associated number of ports that will need the clocked per type
- The goal is to ensure that for each type, the correct # of rows exist in the connectivity section
- If number of rows are less than expected, an error will be issued

1c: Connectivity Definition Table
- Single Table for PAD-IP and IP-IP Connections
  - Table for Mixed connections
  - Table for Non-Mixed connections
- Connectivity Data Entry
  - Manual
  - Through built-in forms
- Debug connectivity failures

2: Formal Verification Flow for Connectivity
1. Capture connectivity specification (spreadsheet)
2. Export .scv, import with RTL to Incisive® Formal Verifier (FV)
3. Generate connectivity assertions
4. Generate black box module list
5. Execute assertions in FV
6. Debug connectivity failures

3a: Basic Connectivity Assertion With Reset
- Black-boxing modules reduces complexity and run-time
  - Requires functional logic that does not affect connectivity from formal analysis
- Manual black-boxing
  - Automatically generates black box for each module
  - Only manual black-boxing was used by this customer
- Automatic black-boxing
  - Incisive® connectivity reuse file
  - Automatically generates black box for each module
  - Incisive® connectivity reuse file

3b: Muxed Connectivity Assertions
- Formal SoC Connectivity flow was easy to setup and use
  - Highly reusable from design to design
  - Verification set-up time reduced to typically 1 day
- Exhaustive verification was achieved
  - With between 0.5-1 full-time verification engineer per project
  - 15 test types completed in under 1 month
- Real bugs found that directed tests would not have caught
  - E.g. black supplied by wrong clock, which was identical to the right clock in all but one corner-case
  - No bugs have escaped, no highly-visible ECO needed, since formal flow adopted

3c: Pipelined Connectivity Assertions

Results and Conclusions
- 3X verification productivity improvement
  - 2 months reduced design time
  - No connectivity bug escapes

4: Debug
- Cross-probing between assertion results in SimVision and connection paths in schematic browser
- Cross-probing between schematic browser and source browser