Framework for Automated Connectivity Checks for core and SOCs

Srobona Mitra, Avinaba Tapadar, Akash Singh,
Mohit Solanki, Raghu B R

Qualcomm India Private Limited

AGENDA

- Introduction
- Motivation
- Formal Connectivity check
- Rules-based connectivity check (static)
- Bugs caught
- Conclusion

Introduction

- Connectivity verification is one of the most time-consuming and crucial tasks due to its volume of checks and the serious repercussions of a bug.
- At SOC level the no. of connections usually goes into lakhs if not more.
- Manually filled in connectivity information are present in IP-Spec releases. An error in this often gets caught at the later stages where it is difficult to check if the bug is present in the spec or the RTL.
- A debugging activity at a later stage requires involvement of multiple teams, namely, Core design, SoC design, and SoC DV.
- This work aims to simplify the prognosis step by *formally* verifying the connectivity of different parts of the design at earlier stages and at core level. We use the Connectivity app of VC Formal in the process.
- We also use rule-based static checks for verifying SOC integration connectivity.
- Our tool has been adopted by multiple teams and applied on several projects.
- It has proven its worth by catching several bugs in these projects thus increasing the quality of verification and bestowing confidence.

Motivation and Our Proposal

Why we need automated connectivity checks

- The data is filled manually !!
- Given the amount of the data on connections, it is highly likely that bugs will creep in during this manual process.
- It takes days to verify the data in DV
- A bug often gets caught in the later stage, where multiple days of effort is required for debugging.

Proposed methodology

- ✓ End-to-end automated
- ✓ Takes minutes to verify
- ✓ Bugs are caught in early stages, easier to debug
- ✓ Can be signed-off without DV

Contribution

We have developed our tool to carry out connectivity verifications both at core and SOC level.

Following is a brief outline of our contribution.

Core level formal verification

- Formal connectivity verification of connectivity data present in spec
- Connectivity gets formally verified within minutes; all core verification can be completed in ~2hours

SOC level static checks

- Developed the syntax and semantics of specifying connectivity rules at SOC level
- Checks SOC level inter-core connectivity based on user-provided rules
- Automated static checks of the SOC-level connectivity
- Provides a SOC level connectivity coverage metric

Rule-based static checks

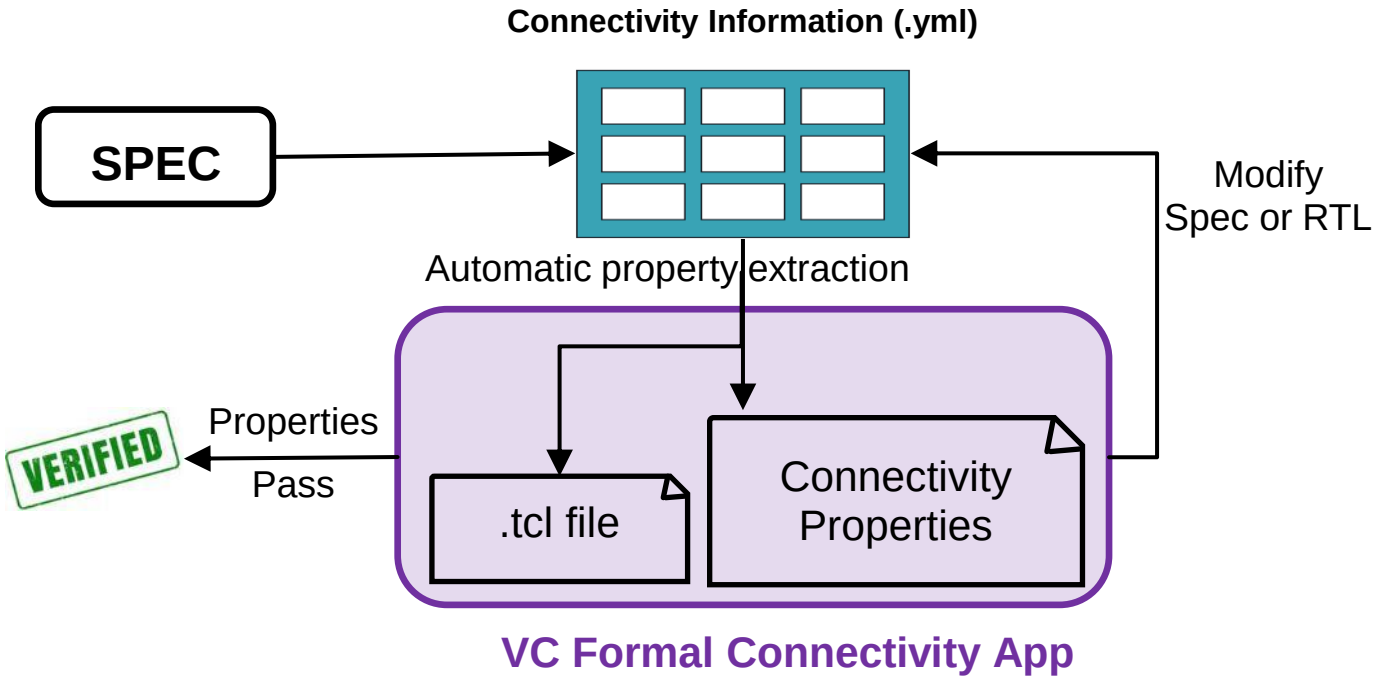
Creating the connectivity rules

- We have developed the following syntax and semantics for specifying the SOC connectivity rules
- It caters the need for customized and varied use cases needed for SOC connectivity checks
- We have introduced two types of wildcard “*” and “+” to denote all matching cases and at least one matching case, respectively.
- There is also provision to provide multiple destinations for a source
- Following are a few examples

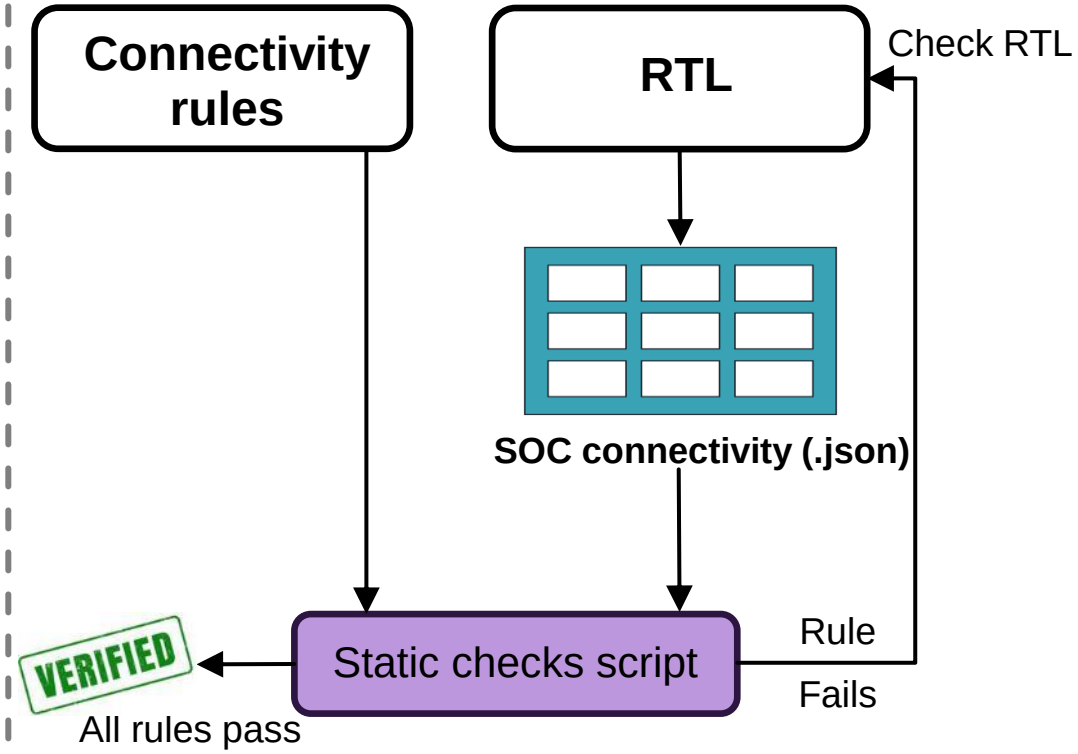
Src instance	Src port	Dest instance	Dest port	Semantics
u_noc	a_irq	u_modem	irq_in	*u_noc.a_irq* → *u_modem.irq_in*
u_noc	a_irq	u_modem		*u_noc.a_irq* → any one port in *u_modem*
u_noc		u_modem	irq_in	any one port from *u_noc* → *u_modem.irq_in*
u_gcc	gcc_apc_clk	u_modem	*_clk	*u_gcc.gcc_apc_clk* → all ports matching with *u_modem.*_clk*
u_gcc	gcc_apc_clk	u_modem	+_clk	*u_gcc.gcc_apc_clk* → at least one port matching with *u_modem.*_clk*
u_gcc	+_clk	u_*	+_clk	At least one *clk* signal from *u_gcc* → at least one *clk* signal in all *u_** modules

Proposed methodology

Core level (formal)



Rule-based static check



Results

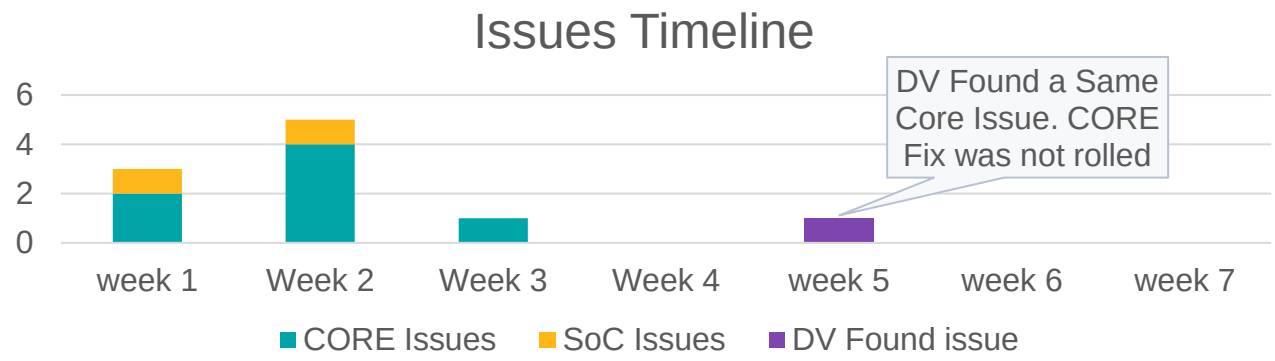
	Traditional way	Proposed way
Core level	• User has to manually generate separate testbenches for checking each memory connection present in IP-Spec. • Verification is done during simulation with manually embedded checkers. • Takes several days	• Push button flow • Minimal manual effort • All cores can be validated in ~2hrs • Bugs caught at early stage of design cycle
SOC level	• No spec present • Takes days to manually verify each connection as per intention • No scope for reusability	• Takes <1hr for entire SOC integration connectivity verification • Spec (rules file) once generated can be reused across projects

Bugs Caught

Our tool has been deployed across multiple projects in Qualcomm Technologies. It has been integrated in the spec-release flow and can be configured to act as a gating check for every core spec release.

Following are some of the bugs which we caught during the initial deployment phase

- Project 1: at least 6 bugs were caught
- Project 2: >50 bugs caught at a very early stage of the design
- Project 3:
  - **Core level** : 9 bugs caught at core-level
  - **SOC level:** Bugs caught at SOC integration level before DV, and patches were provided.



Conclusion and Future work

- If a bug in core spec is caught at a later stage, it halts the further developments until the cause of the bug is figured out and rectified. Since multiple variables come into play at this stage, it is difficult to ascertain the cause of the bug.
- We left shift the verification step to an early stage, hence saving on time and effort needed to debug at later stage.
- Our tool streamlines the connectivity validation methodology with formal checks across different cores and static check in SOCs.
- Our end-to-end automated methodology reduces the verification time from **multiple days** **to** **10 min** (core level) and **1hr** (SOC level) while increasing the quality of the solution.
- The proposed approach has shown promising results on multiple modules across different projects.
- It has been integrated into the spec-release system as a one-click solution. It can automatically report mis-matches between the RTL and spec on every spec release.


THANK YOU
Our Technology, Your Innovation™