Effortless, Methodical and Exhaustive Register Verification using what you already have.

Aishwarya Sridhar

Pallavi Atha

David Crutchfield

accellera SYSTEMS INITIATIVE™

infineon SIEMENS

2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Problem Statement/Introduction

EDA industry provides Formal techniques to simplify exhaustive checking of control and status registers, significantly reducing run time. An example is Questa Register Check which uses the register specification (in CSV or IP-XACT) and RTL as input. Although formal based Register verification provides a significant solution when compared to UVM_REG simulation-based verification, engineers still face following limitations —

1. Nowadays every IP or block in our products has a large number of registers for controlling, configuring, processing, and monitoring tasks such as reading from the IP's processor using the software. As the number of registers increases, writing the input files (register specifications, interface specifications, and run commands) manually becomes complex and error prone.

2. With a rise in Peripheral IP’s complexity, engineers encounter scattered and aliased type of registers where, running the register verification sequentially results in a lot of time consumption.

3. In such scenarios, the output is not consolidated, which makes it hard to analyze the results.

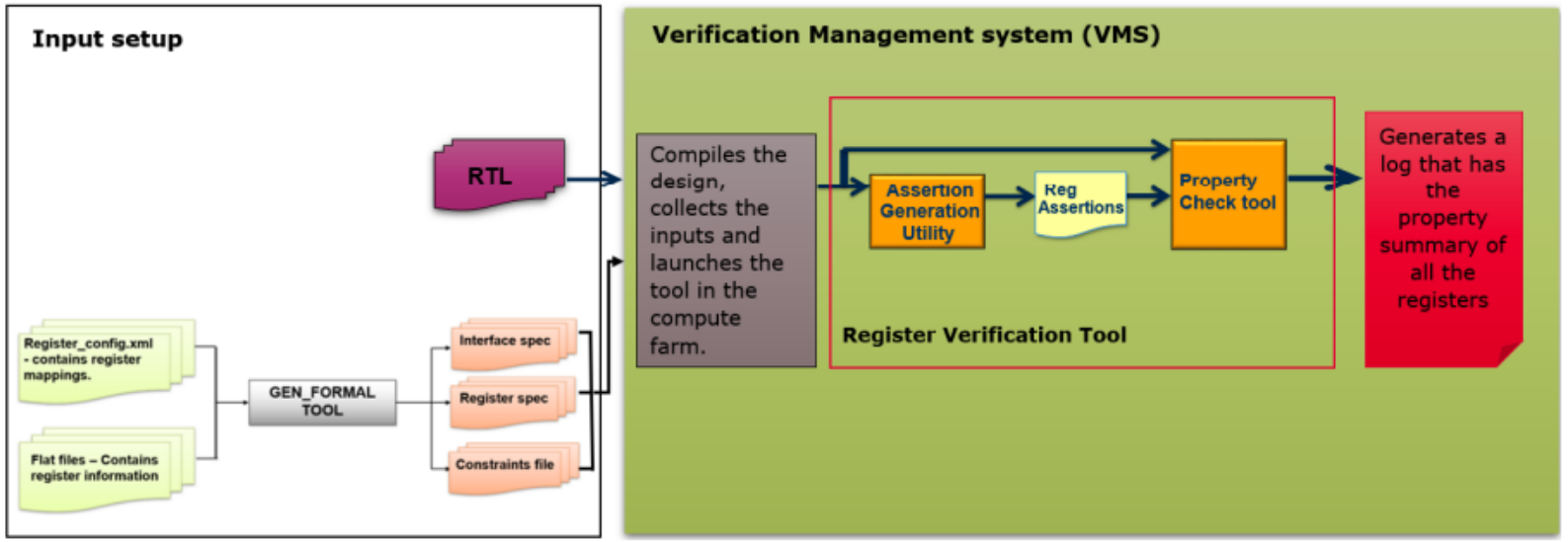
Proposed Methodology/Advantages

To bring a solution, we have developed an automated flow to generate a set of register input configuration files, register specifications, and run commands file using a decision tree algorithm for parsing the IP/SOC flat files; and then automatically run formal register verification on these input files containing the register information. This substantially reduces the required manual effort from a few man-weeks to a few minutes.

The flow then takes advantage of register memory map data, CSV creation, assertion generation, verification plan mapping, and reporting of the property summary that respects the register configurations. Consequently, users no longer have to write the inputs required for the register verification manually which saves a lot of work, time, and cost. Results show significant ROI in terms of quality and productivity.

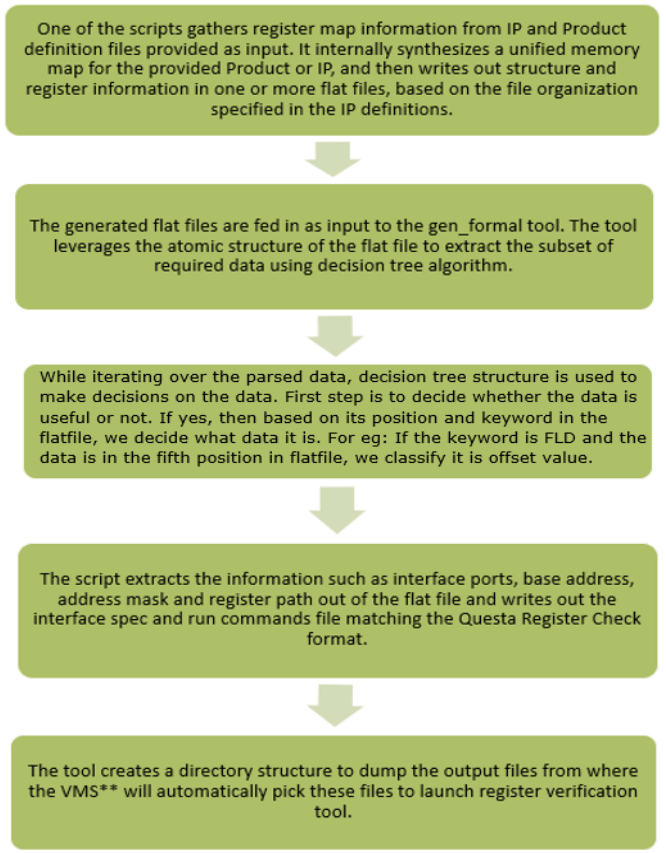
Implementation Details

The below diagram shows the overview of the fully automated, push-button Formal verification flow of the Register check with the “Gen_formal” tool we created. In the setup phase, the RTL design, register config memory map file and IP/SOC flat files are provided by the IP engineers; and the interface spec, register spec and run commands file are auto-generated by the Gen_formal tool.



The outputs of the setup phase are fed into our in-house auto-regression flow (VMS**), which provides a standard scripting environment and acts as an interface to our verification tools. Specifically, this flow collects the input files and user options, compiles the design, and then launches the formal-based register verification app on the compute farm.

Implementation Details



Formal-based verification flow can be run in-parallel across multiple CPUs without any special set-up by the user. VMS** leverages this and it spawns off the register check jobs in parallel across the compute farm, which reduces the users' "wait time" exponentially. Once the verification run is completed, we get one consolidated log with all the needed information.

Results Table

A complex IP with 64 registers had 16 MMIO registers (16 scattered registers) + 48 COMP STRUCT registers (1 scattered register) summing up to 17 scattered registers.

Criteria	Without the automation	With Gen_formal+VMS automation
Input file creation	Created Manually. Being generous and not considering any human errors or error debug time, it would have taken minimum 4 hours on average to create all the input file for one register, then ROI = 17 X 4 hr ROI = 68 hr ROI = ~ 8.5-man days of effort saved	Created automatically in 5 seconds.
Register check run time	Register check ran sequentially which took 18 minutes	Register check ran parallelly which took 5 minutes
Output	17 output files.	One consolidated output file that has all the 17 registers' property summaries.
Overall Time taken	**8.5days + 18minutes**	**5minutes + 5seconds**

NOTE: The calculation for the amount of time saved in creating the input files for formal register check using Gen_formal automation flow, where:

N = the number of scattered registers

T = the time it would have taken you to manually create the input files

ROI = total amount of time saved automating the input generation for register verification (N x T)

Conclusion

The Gen_formal + VMS automation is 1.2 x 10^3 times faster, thus saving considerable amount of time and reducing manual efforts. Using this flow, engineers can just focus on catching the register related bugs early on by leveraging:

- Exhaustive checks access of control and status registers.
- Register address correctness checks.
- The default value on resets checks.
- Stability of register data checks.
- Verification of expected behavior and absence of illegal behavior.

We intend to use this automation for all formal based register verification of suitable designs in the future.

REFERENCES

- **[1] David Crutchfield, Thom Ellis, “Bringing Regression Systems into the 21st ”, DVCon,2014.

- [2] Lee Burns, David Crutchfield, Bob Metzler, Hithesh Velkooru,” Using Formal Applications to Create Pristine IPs”, DVCon, 2017