A Structured Approach to verify Ties, Unconnected Signals and Parameters

Saurabh Singh, NXP, Munich, Germany (saurabh.singh@nxp.com)

Peter Limmer, NXP, Munich, Germany (peter.limmer@nxp.com)

Thomas Luedeke, NXP, Munich, Germany (thomas.luedeke@nxp.com)

***Abstract*— The paper presents a structured approach to verify ties, unconnected signals and parameters in a design that replaces the traditional mixed approach of visual inspection, directed tests and formal verification [1]. The approach requires no prior knowledge of formal tools and languages like C, SystemVerilog and leads to comprehensive verification, improvement in quality of documentation and significant reduction in debug time.**

Keywords—RTL design; RTL verification;

I. Introduction

Automated checks is the mechanism used to implement the proposed approach. The biggest motivation for automated checks is to replace visual checks done for ties, unconnected signals and parameters. The visual checks, done by looking at the simulated waveform, are not regularly done for every RTL release made by Front End teams. While some ties and parameters are checked by writing test cases or formally using tools like Cadence Incisive Formal Verifier, a consistent structure is not adopted by everyone in the team [2-3]. There is also no regular mechanism to check unconnected outputs.

The developed and deployed approach of automated checks is done for every RTL release and hence catches incorrect ties, unconnected signals and parameters (henceforth called TUP. T = ties, U = unconnected signals, P = parameters) early in the verification phase. They also catch any unspecified TUP in the integration documents/guides, leading to improvement in documentation. The checks catch any RTL change with respect to TUP and can run independently or with regression.

II. DESCRIPTION

A. Methodology

Since TUP are structural properties, a structural tool like Cadence Encounter Conformal Equivalence Checker is used to extract information about TUP present in the RTL. The information database thus obtained is the Front End (FE) database. The FE database contains information about TUP for every instance in the RTL. The database is split in three separate parts - FE tie database, FE unconnect database and FE parameter database. The three FE databases are compared respectively against the three Verification databases – Verification tie database, Verification unconnect database, Verification parameter database. A Verification database contains three separate text files for TUP for every instance in RTL. The information for the Verification database is obtained from the integration guides which state the TUP expected for every instance present in the design. Verification databases are updated manually and not extracted through scripts from integration guides. Therefore, if an update is made both to RTL and an integration guide, the changes in integration guide do not automatically propagate to the Verification database. This helps Verification database to catch any update made to RTL. Figure 1 represents the whole concept.

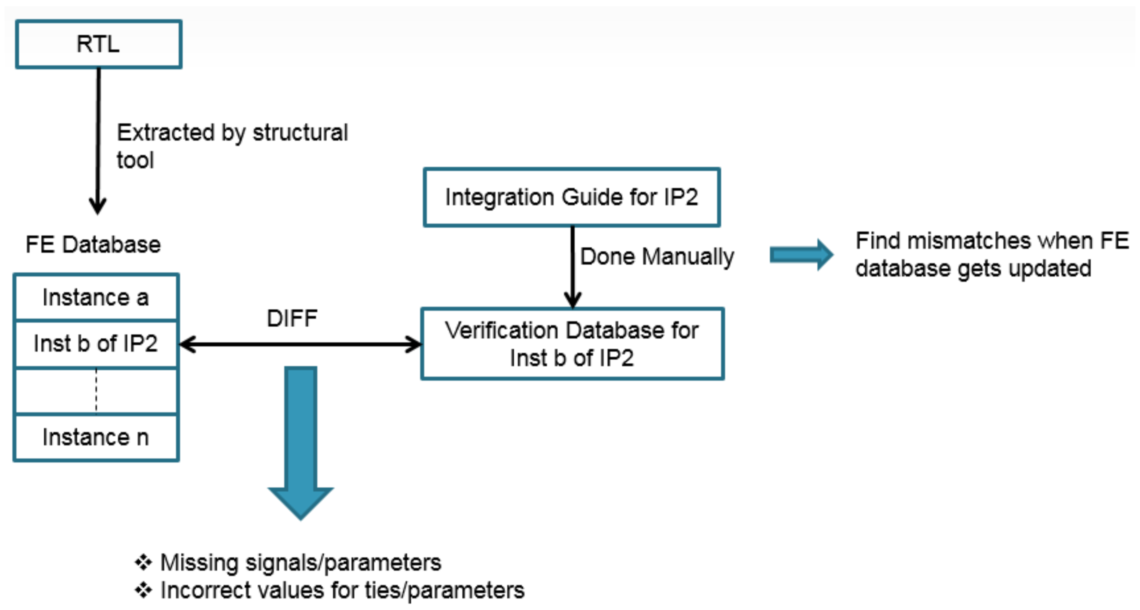
RTL
Extracted by structural tool
FE Database
Instance a
Inst b of IP2
Instance n
DIFF
Integration Guide for IP2
Done Manually
Verification Database for Inst b of IP2
Find mismatches when FE database gets updated
❖ Missing signals/parameters
❖ Incorrect values for ties/parameters

Figure 1: Automated checks methodology

When the FE tie database for instance n, for example, is compared against Verification tie database for instance n, mismatches (if any) are highlighted between the two databases. In general, mismatches for an instance n can be of three types:

- TUP present in the FE database and missing in the Verification database. This means the RTL has TUP which are unspecified in the integration guide.
- TUP missing in FE database and present in Verification database. This means that integration guides lists some expected TUP which are missing in RTL.
- Value mismatches in case of ties and parameters. This means that ties and parameters are present in both FE and Verification database but their values don’t match.

Mismatches indicate either a bug in RTL or inaccuracy of integration guide.

B. *Input Format*

Input format indicates the format in which the expected TUP are written in the Verification database. The format is human and machine readable. The input format for a particular signal/parameter has the following sub-parts:

- Signal/parameter name
- Width: gives flexibility to the user to check the whole multi-bit signal or a part of the signal.
- Data type of tied/parameter value: pseudo Verilog style is adopted. This can be hexadecimal, decimal or binary.
- Value in case of ties and parameters.

Figure 2 gives an example of input format for a tied signal of an instance n, written in three different ways by choosing different data types. Figure 3 gives an example of input format for unconnected signals of instance n.

```
ipp_ind_pqspi[11:4] 8 d 0
ipp_ind_pqspi[11:4] 8 h 00
ipp_ind_pqspi[11:4] 8 b 0000_0000
```

Figure 2: Input format for tied signal ipp_ind_pqspi of instance n. Bits 4 to 11 are tied.

```
ipp_ana_en_sample[15]
ipp_decode_extch[2:0]
```

Figure 3: Input format for unconnected signals of instance n

C. Output Format

The output format refers to the format of log files obtained by comparing the FE database against the Verification database. The log files indicate whether the comparison is successful or unsuccessful and highlight the mismatches if comparison is unsuccessful. The mismatches are grouped in three sections: first section indicates TUP missing in FE database (if any), second section indicates the TUP missing in Verification database (if any) and the third indicates the value mismatches (if any) in case of ties and parameters. Figure 4 shows the output format in case of mismatches between FE tie database and Verification tie database for instance n.

```
Instance name: instance_n

SIGNAL(S) MISSING IN FRONTEND DATABASE
Signal Name : signal_a
Signal Name : signal_b

SIGNAL(S) MISSING IN VERIFICATION DATABASE
Signal Name : signal_c
Signal Name : signal_d

VALUE MISMATCHES
Signal Name : signal_e
Value in frontend db : 3 d 1
Value in verif db    : 3 d 2

Signal Name : signal_f
Value in frontend db : 8 h 1
Value in verif db    : 8 h 5

SCRIPT_FAIL: 6 Error(s) detected !
```

Figure 4: Output format example in case of mismatches for instance n.

III. Results And Conclusion

The developed method has been successfully deployed on two 32-bit SoCs. FE database is generated once for a given RTL database and takes around 4 minutes on the pilot project. Verification database is created once per project and can take from few seconds to few minutes per instance depending its complexity. Updating Verification database and running FE and Verification database comparisons just takes few seconds resulting in very fast turnaround time. Automated checks compare RTL and documentation in both directions. The approach found new unknown TUP in addition to inaccuracies in the form of value mismatches, incorrectly expected TUP in RTL and documentation. Formal tools don't find unspecified and unknown TUP. Structural verification also has a requirement for documentation and tagging. With the help of automated checks, the quality of documentation has improved. The approach is an important and efficient complement to formal and simulation-based verification environments.

The test creation, run and debug times are very less compared to directed tests or formal verification. There is no need to get the simulation up and running. The checks are easy to adapt and the result is shown in an easy to understand format. There is no need to know any tool like Cadence Incisive Formal Verifier or languages like C and SystemVerilog. While creating the checks, time is also spent on validation of parameters. The checks help to kick off the project. There is no need to testbench infrastructure for verifying TUP. Exceptions for toggle coverage are also generated from the same flow.

REFERENCES

[1] R. Drechsler and G. Fey, “Formal Verification meets Robustness Checking – Techniques and Challenges,” in Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on, pp. 4-4, April 2010

[2] P. Tiwari, R. Mitra, M. Chopra and A. Jain, "Formal Assertion based Verification in Industrial Setting” in 44th Design Automation Conference (DAC), June 2007, URL: http://www.facweb.iitkgp.ernet.in/~pallab/mitra_Tut3_v3.pdf, URL accessed on: 19.07.2016

[3] G. Zhong, J. Zhou and B. Xia, "Parameter and UVM, making a layered testbench powerful" in ASIC (ASICON), 2013 IEEE 10th International Conference on, pp. 1-4, October 2013