SoC Functional verification flow

Dilip Prajapati - December 14, 2017

This paper presents SoC- (System on Chip) level functional verification flow. It also describes ways to speed up the process.

To ensure successful tapeout of SoCs, here are the steps of a standard SoC-level Functional Verification flow:

1. SoC Level/Top Level view (Feature Extractions)

During SoC verification, you must view the design at the top level and extract its SoC level functionality/features during specification study phase for its verification. At this stage, a thorough understanding of SoC functionality and its architecture is required because misunderstanding of the specification can become the leading cause of bugs, and due to this you may waste unnecessary time on issues which are not real RTL problems.

2. SoC Level Verification Plan

- Define a Clear Line Between SoC and IP

During the development of the SoC level verification plan, you have to clearly define/identify the functionalities, which needs to be verified at the SoC level and at the sub-block or sub-IP or sub-cluster level. The same verification needs to be confirmed during the review with the respective verification team to avoid any last-minute surprises.

- Identify Reusability Components

You must also check which block or sub-block level verification components/environment and scenarios can be reused at the SoC level with a reusability point of view to reduce its new development time at SoC level.

- Verify Interconnections

At SoC level, you mainly have to focus on the top level functionalities of the SoC along with verifying whether the intercommunication between the sub-blocks occur properly or not. You have to also verify the connection of the sub-modules with the top.
• Keep Placeholders for Updates

Sometimes, all the features of the SoC are not defined at the initial phase of the SoC verification planning. For those types of features, you need to update the verification plan at a later stage. Hence, during verification plan development, you can put placeholders or FIXME for the same to update the features when the need arises later on.

SoC-level functional verification flowchart

3. SoC Level Verification Environment and SoC Verification

The SoC level verification environment contains sub-block level verification components/environment along with dedicated SoC level verification components/environment.
• **SoC Level Verification Methodology**

For the SoC verification, you have to choose the right verification methodology which is best suited for its thorough verification. Along with this, you have to also check in terms of its reusability and its compatibility with the legacy verification environment, if it is available and if you are going to reuse the same.

For SoC Verification, either a mix methodology/mix languages-based verification environment or a single standard methodology-based environment is selected based on the SoC’s functionality.

• **Integration of Sub IPs/Blocks/Modules/Clusters**

Before the actual SoC verification starts, the first step is to integrate/stitches of the sub-blocks/sub-IPs/sub-clusters into the SoC level verification environment. This is one of the major activities of SoC verification. You also have to develop connectivity checkers which will make sure whether the integration of the sub-blocks to SoC is done properly or not. This will play a critical role in SoC level verification because if the integration is not done properly, then you cannot expect SoC level targeted functionality out of it.

• **Legacy Environment/Reusability and New Verification Environment Development**

As a part of reusability and speed of the verification process, first check whether there is a generic legacy verification environment available, which you can reuse for verification. For example, if you are working on SoC, check if it is the enhanced version of its previous SoC with some additional functionality and enhancement in it. For this, you can reuse its legacy environment and enhance it to verify the updated functionality in it. You can also reuse sub block’s Monitors, Checkers, Assertions and Reusable/Scalable Test Case Scenarios at the SoC level.

As a part of SoC Verification Environment development, you need to develop standard verification components like SoC level Driver, Monitors, Scoreboards/Checkers/Assertions, Reference Models, Functional Coverage and Top level Test Bench. For processor-based SoC, generally, a processor is used to configure the whole SoC/its sub-modules and driving the SoC level stimulus. For standard protocols, (like PCIe, Ethernet, USB etc.) you can also use third-party VIP (Verification IP) available in the market to speed up the verification.

• **Verification of the Analog Macros at SoC level**

For analog macros, an equivalent digital reference model from the designer for the SoC level functional verification is available because, at SoC level, the analog mix-signal simulation takes more time. Generally, an IP level team, which owns the respective analog macros will ensure its functionality with analog mix-signal simulation during its verification at IP level.
SoC Level Scenarios

At SoC level verification, you may require developing SoC level scenarios to verify its functionality at the top level with an end user's point of view. If the IP level team has developed reusable/scalable SoC level test case scenarios then you can reuse the same at SoC level which will help you speed up the SoC verification. You can also reuse the legacy test case scenarios with some modification for the similar functionalities of the SoC which were present in its previous version.

SoC level Debugging

Debugging is a time-consuming process, especially when it comes to the complex SoC verification. A proper understanding of the detailed architecture or the blocks which you need to verify at SoC level will help you reduce the unnecessary debugging time to a great extent.

Use proper coding style and smart message printing with user control verbosity, switch ON the required sub IP level messages, enable debug switches of the verification methodology and use proper debugging features of the EDA tool which will help you to quickly debug of the RTL as well as the test bench/VE (Verification Environment) issues at SoC level. For example, if you are using UVM, then you can trace the objection activities using +UVM_OBJECTION_TRACE plus an argument on the command line.

During bug filing or any RTL issue reporting, you should provide all the minor details, including the steps to reproduce this issue and the waveform, if applicable to the designer will help him to fix the issue quickly!!!

4. Functional and Code Coverage Closure

Functional and code coverage closure is one of the major milestones for the successful tapeout of SoC. Proper analysis and review of the functional/code coverage will help you close it to 100%. Generally, at SoC level, cumulative regression results/merging of the consecutive regression results are used to close the functional and code coverage. In the code coverage for the uncovered code, you should have proper justification as to either it is redundant RTL code or that code will be never exercised at the SoC level or it is not covered during functional verification. If the code is not going to be covered at the SoC level or not used for this SoC, then those types of codes have to be added to the exclusion list after designer's confirmation.

5. Final Functional Verification Closure (Verification Sign-off)

For the final functional verification closure, you have to fulfil all the functional verification criteria that have been defined in the functional verification sign-off list. For example, 100% functional and code coverage, consecutive defined number of clean regressions, all bugs closed/fixed, etc. Functional and code coverage will provide confidence on the maturity of the design and its functional verification closure.
Related articles:

- Guidelines for complex SoC verification
- Reduce SoC verification time through reuse in pre-silicon validation

—Dilip Prajapati is an ASIC engineer at eInfochips.