Front-End Universal VLSI Design Verification Methodology using System Verilog (SV) Architecture for Design Verification Engineer

Amay Shiva Naik[1], Feroz Ahmed Choudhary[2], Deepa M K[3], Bharathi S H[4]

[1,2,3,4]Dept. of Electronics and communication Engineering, REVA University, Bengaluru, India

***

Abstract – *VLSI industry or exposure can be classified in two parts distinctively which are Front-End and Back-end. This paper mainly deals in Front-End part for Design Verification Methodology which will give Basic understanding of Universal System Verilog Architecture which can be used for any Design Verification.*

Before going to VLSI concepts we need to understand the difference in Hardware and Software languages which will in turn answer why HDL is for Designing and Verification. HDL have options such as VHDL, Verilog, System Verilog etc. where Verilog is preferred for writing Top module Design whereas System Verilog is used for other Testing Blocks.

***Key Words*: VLSI Design Flow, Front-End VLSI, Back-End VLSI, System Verilog (SV) Architecture, Verification Scenario, QuestaSim Tool etc.**

1. INTRODUCTION

The base of Front-End depends on HDL, which leads to one simple question in every mind why we can't use Software languages for writing codes for Hardware Design & its Verification. The answer to that question is very simple is that Software Languages does not have the concept of Time Constraint thus make it inappropriate to be used in place on HDL.

HDL has many options to choose from namely VHDL, Verilog, System Verilog etc. Most of times a coder prefer Verilog over VHDL mainly because of improvement as well as Verilog feels easy for learning and grasping the basic contents which is required for coding. System Verilog can be called as a hybrid of C, C++ and Verilog i.e. it includes the concepts of Verilog and the Object Oriented Programming (OOPs) concepts of c, c++. SV is preferred for writing codes for blocks required in Verification and Testing whereas Verilog is used for writing code for Top module Design.

Firstly lets understand the basic role play of each i.e. Front – End & Back-End in VLSI domain. VLSI Design Flow (Fig 1.) Explains all partly role played by both Ends.

1.1 VLSI Design Flow

As can be seen in fig 1 first 4 blocks represents role of Front-end and the remaining blocks represents the role of Back–end. Front-end Engineers job is till RTL synthesis after that for the manufacturing of that Design/Device is done by Back-End Engineers.

Let us understand the flow by taking an example say a client approaches a Front-end Engineer with a Design Specification, Front-end understands the client's requirement and writes the top level code for the design in any Advanced Verification Tools. The top level code needs to be validated i.e. verified by using SV architecture after getting the results is the design is acceptable its RTL synthesis is given to the Back-end Engineers. The tool required for the back-end are different from those for Verification which are mostly deals in Placement & Routing of cells for the device it also involves Verification which different from Front-end Verification.

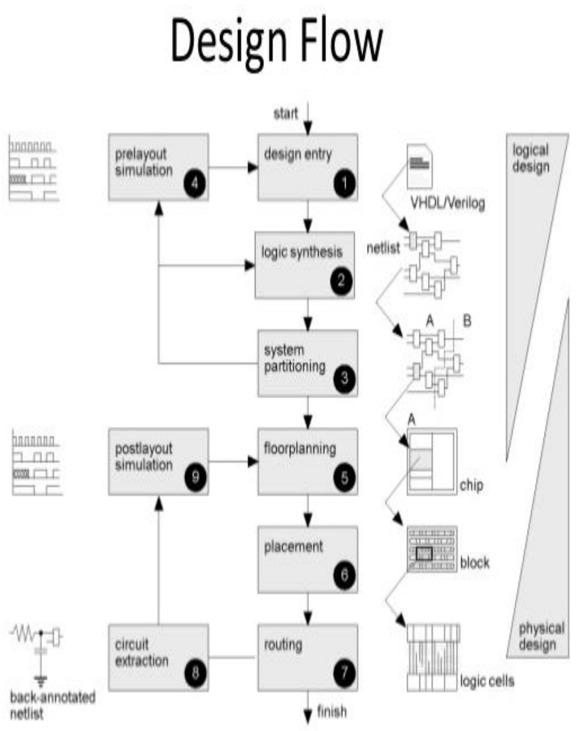

Fig -1: DESIGN FLOW

Verification in Front-end involves writing Test cases whereas in case of Back-ends is verifying the cells order placement for manufacturing purpose.

2. System Verilog Architecture

The below show Fig-2 is for the System Verification Architecture as can be seen there are various blocks which are need to coding in the tool, but the question is why Verilog only can be used and why in what way does System Verilog have advantage over Verilog. DUT Block is the Device under test i.e. the top module for which coding is done by using Verilog. If all the remaining Blocks are coded via Verilog then we have to instantiate coding by defining module name for each block, module to module communication is very hectic thus we don't prefer Verilog for coding of other blocks instead prefer System Verilog. The main reason is System Verilog includes OOPs concepts thus defining each blocks codes in a class format provides easy way of coding as compare to Verilog.

The following Figure can be divided as two parts for master and slave configuration, the left hand side is for master and the right hand side is for the slave. Depending upon the no of slave and master the figure can be modified for different design aspects.

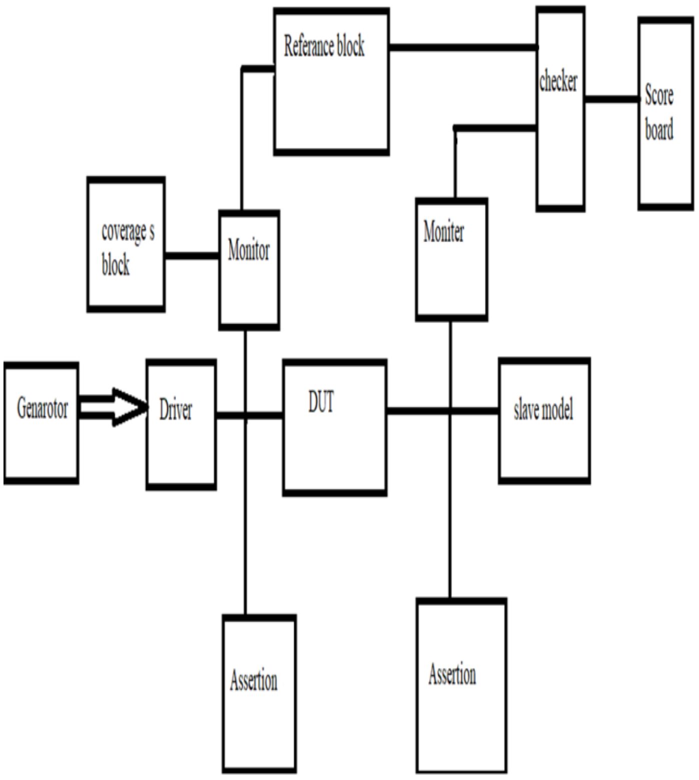

Fig -2: System Verilog Architecture

We can briefly define each block role independently as follows

(i) Generator Block: Responsible for generating all the signals required for all scenarios example clock & reset signals etc.

(ii)Driver/Bus function model (BFM): Routing Signals generated from the generator.

(iii)Monitor: To check signals and has inputs from coverage & assertion blocks.

(iv)Coverage Block: checking the given input to the DUT whether it's complete or not.

(v)Assertion Block: checking at the interface as per the protocol or not i.e. indicates violation of protocols.

(vi)Reference: kind of having desire results.

(vii)Checker: Compare the outputs from the reference and DUT block.

(viii)Scoreboard: Displays the result of Checker.

3. Verification Steps

(i) Features listing down.
(ii) Scenario listing down.
(iii) Test plan development.
(iv) Functional Coverage Point listing down.
(v) Testbench architecture definition.
(vi) Testbench component coding.
(vii) Sanity test case development.
(viii) Sanity test case bring up.
(ix) other test cases.
(x) Setting up regression.
(xi) Running regression and debugging regression results.
(xii) Generating coverage results.
(xiii) Analyze coverage results.
(xiv) Closing functional Coverage.

4. Verification Tools

There are many companies which provides simulating advanced verification tools namely QuestaSim, ModelSim, Xilinx, Cadence etc.

Usually the tools works in 3 steps which includes:

(i) Compilation.
(ii) Elaboration.
(iii) Simulation.

Tool like QuestaSim has a inbuilt writing notepad otherwise the code written in a separate notepad can be linked to the tools. For every Design prospective we need to create a new project in the tool and make sure all the required libraries linked to the directories are included for the same project.

5. Conclusion

Universal Verification Methodology was studied for verifying any design. The aim is to get 100% Functional Coverage results i.e. verifying all test case scenario in order to get better quality design for manufacturing purpose.

REFERENCES

1. System Verilog for Quality of Results (QoR) **DOI:** 10.1109/ISQED.2008.4479777.

2. A loosely coupled C/Verilog environment for system level verification. DOI: 10.1109/IVC.1998.660697.

3. 1364-1995 - IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language. DOI: 10.1109/IEEESTD.1996.81542.

4. Guide lines for safe simulation and synthesis of implicit style Verilog.
DOI: 10.1109/IVC.1998.660681.