Some aspects of System-Level Verification of ASICs/FPGAs

Introduction

Over the years, chips have grown in size and complexity. It is common for chips to contain one or more embedded processors running firmware, multiple embedded RAMs, and multiple design IPs (intellectual property blocks) from different IP vendors.

As chips grow in complexity and more highly integrated, functional verification will require a more systematic approach to ensure that the chips are verified and completed within a reasonable time.

This paper discusses some aspects of system-level functional verification of ASICs and FPGAs.

Choosing Verification Methodology

Most front-end RTL designs are still coded either in Verilog or VHDL hardware description languages (HDL). The subset of HDL language syntax used for design coding has not changed much over the years. By contrast, HDL language features and methodologies for verification have developed extensively over the last fifteen years.

For current chip development, the main languages used for verification are SystemVerilog and UVM (Universal Verification Methodology). SystemVerilog is a major extension of Verilog with many added features, including constrained randomization, functional coverage, assertions, and enhancements to the various features of the original Verilog language. UVM is developed on top of SystemVerilog to provide class-based verification with standard base classes and methodology.

For the verification tasks, the chip development team will have to decide whether to use SystemVerilog only, or SystemVerilog in combination with UVM. At a minimum, any current chip project should at least use SystemVerilog for verification instead of the older Verilog or VHDL.

Whether UVM should be used depends on a number of factors that are unique to the specific requirements of the project. These include the nature of the chip design (e.g. using standard protocols, multiple masters and targets, etc), whether UVM VIPs are available or developed internally, the experiences of the team, and other factors unique to the chip project. UVM is complex and it takes some experience to learn how to use its features effectively for verification.

Using Vendor IPs

Chips with integrated third party design IPs (design Intellectual Property) are now common, especially chips that use industry standard interfaces such as PCIe, USB, and so on.

These design IPs would have already been fully verified at the block level by the IP vendors. After integrating the IPs into full design, verification is required at the system-level to ensure that the chip design works according to specification.

To assist in verification, some design IP vendors may provide corresponding verification IPs (VIPs) for simulation. Other vendors may only provide a base verification IP and the team may need to purchase VIPs from other third party vendors, or else the team will need to develop them internally. The vendors may have both a non-UVM and a UVM version of the VIP.

Preparing Testplan

The testplan will include one or more documents that describe all the test cases to be run to verify the DUT.

To prepare the testplan, the verification engineer must read and understand various specifications, including internal architecture and design specifications, vendor design and verification IP specifications, industry standard protocol specifications (e.g. PCIe, USB, I2C, etc), and others as required.

A thorough understanding of the various specifications is required to come up with a complete testplan and to create good test cases to 'break' the DUT during simulation. Typically, the test cases will range from simple directed tests to testing complex corner case scenarios. The complex test scenarios include generating random stimulus, simulating asynchronous events, interactions among multiple concurrent design blocks, and so on.

Creating Verification Environment

To run system-level simulation, bus functional models (BFMs) and/or vendor VIPs will be used to generate and observe stimulus to the design under test (DUT). The team will typically need to develop some new BFMs in addition to any vendor VIPs.

A system-level testbench is created to verify the DUT. The testbench will include one or more instances of the DUT(s), bus functional models (BFMs), vendor VIPs, and any initialization code. The test cases generate stimuli and transactions to the DUT via the BFMs or VIPs. The DUT is then verified that its behavior meets the chip specification.

Verification includes checking correct interface protocols, correct end-to-end data transfers, and other cases. Interface protocol verification is usually performed by the BFMs or VIPs. End-to-end data transfers are usually checked via scoreboard models. In more complex scenarios, the test cases will initiate multiple concurrent transactions or threads to stress the DUT, and to auto-check the correctness of data transfers and protocols.

The testbench should be developed to allow simulation of the DUT under different design configurations, e.g. different bus width, clock frequencies, operation modes, etc. For ASIC development, another consideration is to allow simulation of the DUT at different levels of design representations, such as RTL, post-DFT, back-annotated gate-level, etc.

Embedded Firmware Co-Simulation

Many chips have one or more embedded processors running firmware to control the functions of the chip. Embedded firmware is typically developed in C, C++ or equivalent. The compiled binary code can be converted to a hexadecimal or binary text file for use in simulation together with the DUT. This is known as hardware-software co-simulation.

In co-simulation, the hexadecimal text file is loaded directly into the processor memory model in simulation. When running the test cases, the embedded processor will be executing firmware and concurrently the BFMs and VIPs will be generating transactions. In this way, the interaction between firmware and the various design blocks can be simulated. This is an effective way to catch firmware and design issues prior to chip tapeout.

Code Coverage and Functional Coverage

Code and functional coverages provide some measures of the thoroughness of the simulated test cases. Code coverage and functional coverage are not the same.

Code coverage is typically run for block level simulation. It is usually not run at the system-level due to the huge effort to analyze code coverage results of the entire chip. The desired coverage for the block is usually set at 100%. If this is not achieved, the coverage results are analyzed and then more test cases can be developed to achieve 100% coverage.

Functional coverage is user-defined and can be performed at the block or system level. Functional coverage fits well into random simulation where the test vectors are randomly generated. Functional coverage can provide feedback as to how thoroughly the random test vectors has exercised the design block.

It is noted that both code and functional coverages are supplements for the main testplan. Achieving 100% code coverage does not guarantee full functional correctness of design; it just means that the test cases touched and toggled every line of the design code, but it does not indicate full functional correctness of the

design code.

Likewise, achieving 100% functional coverage is not the same as achieving full functional correctness of the design. Functional coverage is user-defined, and a 100% functional coverage means that whatever the engineer has defined for coverage is 100% exercised by the test vectors. Functional correctness (meeting protocol specification, end-to-end data checking, etc) of design will have to be performed and checked by the test code, BFMs, scoreboards, and so on.

Formal Verification

Another new verification technique is formal verification. At the present time, formal verification can only be run at the block level, as the tool will not converge when analyzing the huge system-level chip design space.

A formal verification tool requires the verification engineer to specify the design behavior using SystemVerilog assertions. It is crucial that the full set of assertions that fully describe the block design is specified by the engineer. A 100% proven design is only with respect to the set of user-specified assertions. If the engineer inadvertently misses out specifying one or more assertions, the formal tool will be unaware of this and will only prove the design based on an incomplete set of assertions. In this case, a formal tool reporting 100% proven design may not mean that the design is 100% functionally correct.

Managing Design and Verification

A chip project involves a group of engineers in different technical areas of design, verification, backend, testing, firmware, etc. Some projects have engineers in multiple remote sites. It is highly desirable for the group of engineers to work on a common database that can keep track of all work and changes to all project files. A common tool used for this is CVS.

System-level verification requires running many test cases. For large chips, there may be hundreds of test cases to run, and so a regression run strategy should be put in place. Scripts can be developed to initiate running all the tests or selected groups of tests in batch mode without user intervention. At the end of the run, the scripts can then automatically report the results of the regression runs.

FPGA Prototyping

FPGA runs orders of magnitude faster than simulation, and this can provide a fast feedback as to whether something is working or not.

If the design is simple, FPGA prototyping may be able to fully verify the design. However, for most current chips, FPGA prototyping alone will not be able to fully exercise all possible scenarios. By contrast, functional simulation has greater control and freedom to create test scenarios using VIPs and BFMs, including generating random stimulus, creating concurrent events, controlling interaction among blocks, and so on. Simulation also allows the internal signals of the DUT to be controlled and observed for debugging, which are harder to do in FPGA prototyping. Simulation can also provide code and functional coverages.

FPGA prototyping can be a good supplement as part of system-level verification. For most current chip designs, it is essential to have a good simulation strategy with a thorough testplan to ensure catching as many issues as possible prior to tapeout.

Conclusion

As chips grow in complexity and highly integrated, more effort is now focused on functional verification. This article discusses some aspects of functional verification of ASICs/FPGAs.

Author

Chang Chee Keng, Sys-ASIC Designs Pte. Ltd.

C.K. Chang, B.S.E.E., M.S.E.E., has 30 years of experience in the chip development industry in Asia and USA, six of which are in Silicon Valley. At Sys-ASIC Designs, he provides consulting and training with a focus on system-level verification, front-end design and methodologies.

Before Sys-ASIC Designs, he was a technical staff at AMCC/JNI Corporation (Singapore) and participated in architecture, design and verification of 10Gbit Fibre Channel host adapter ASICs. Prior to that, he founded a company to provide consulting and training services to companies in Singapore and the region. Previous to that, he was a Senior Staff Engineer at Silicon Systems (Singapore), a subsidiary of Texas Instruments Inc, and was involved in the design of Firewire storage integrated circuits.

Prior to returning to Asia, he worked in Silicon Valley, California, USA at ESS Technology, 3DO Company and Sun Microsystems in architecture, design and verification of various ASIC development projects.

Mr. Chang has experience and knowledge in industry standard protocols including PCIe, SATA, SAS, Fibre Channel, USB, IEEE 1394 (Firewire), ARM/AMBA, I2C, SPI and JTAG 1149.

Contact

Sys-ASIC Designs Pte. Ltd.

21 Bukit Batok Crescent,

#09-79, WCEGA Tower

Singapore 658065

Tel: 6022 1812

Email: ckchang@sys-asic.com

Web: www.sys-asic.com