International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015
ISSN: 2395-3470
www.ijseas.com

Optimization of ASIC Design Cycle Using Different Verification Techniques

Maitrey Patel[1], Chintan Raval[2], Bhargav Tarpara[3]
[1, 2,] *M.Tech., VLSI, U.V.Patel College of Engineering and Technology, Kherva, Mehsana, India*
[3]*Verification Technical Assistant, eiTRA, Ahmedabad, India*

***Abstract*—From the last 40 years, Integrated Circuit (IC) complexity has increased drastically. By complexity, we refer here to the number of transistors that can be integrated on a single chip. As per the above mentioned statements the design complexity also increases as the transistor count increases but size of transistor decreases and more functionality gets added to chip in the same space or in the reduced space. As the functionality increases the time to verify the design also increases. With the increase in time to verify the design the market demand is increased to reduce or to optimize this time and there are various methods involved in this verification. In this paper, we have tried to describe various ways comparing their effect on verification time and complete design cycle, with the conclusion of selecting modeling as better mechanism.**

Keywords—*RTL (Register Transfer Level), ASIC (Application Specific Integrated Circuit), FPGA (Field Programmable Gate Array) API (Application Programming Interface), DUT (*Design under Test*), ABV (Assertion-based Verification, TLM (transaction Level Modeling)*

I. Introduction

The complexity of today's Systems-on-Chip has increased drastically, making the use of RTL (Register Transfer Level) design methodologies time consuming and error prone [1]. The ultimate goal of ASIC verification is to obtain the highest possible level of confidence in the correctness of a design. With increases in complexity and gate count of an ASIC design, functional verification has become one of the greatest concerns of design engineers. Verification has also become a serious bottleneck in the VLSI design process. This dual challenge of increasing complexity and decreasing time is creating an urgent need for the application of advanced verification methods [2] [3].

In these paper we have divided some verification methods which optimizes ASIC verification into Hardware Based Method giving review about Emulation and FPGA prototyping, Re-usability Method that deals with verification environment reuse, Abstraction-based Method by making a level above RTL called transaction level, Assertion Based Method, Co-verification based method by creating parallel hardware & software execution environment and finally, the Model-Based Methods which deals with creation of models of important system components[3] .

The remainder of the paper is organized as follows. Section II describes various ways which optimizes ASIC design cycle timings.

II. Methods of Optimization

A. Hardware Based Method

The first approach to control the verification bottleneck is to go for Hardware based methods. The three main options are simulation, emulation, and FPGA-based prototyping [3].

In software simulation based verification, the HDL code of the digital logic is simulated by the simulation software. Logic simulation is the primary tool used for verifying the logical correctness of a hardware design. In many cases, logic simulation is the first activity performed in the process of taking a hardware design from concept to realization. Test-bench can be written around the design under test

(DUT) and inputs can be passed to the DUT through the test-bench. Simulation is completely generic and any hardware design can be simulated. Setup is simple, quick and easy highest level of controllability and observability Designer gets complete feedback of the verification process [2] [3].

The ease of software simulation based verification comes with an overhead of simulation time which increases with the complexity of the design. Time consumed in simulating a digital design is a major drawback of simulation based verification. The time required to verify the design is proportional to the maturity of the design. Early in the design process at module level verification, incorrect functionalities are usually found quickly and easily through simulation. As the design matures, it takes longer to find the errors. Simulation time largely varies depending on the software used, computer configuration and coding style. It runs six to ten orders of magnitude slower than the actual ASIC hardware, which makes it an extremely time consuming and inefficient technique [2] [3].

Emulation based verification is a faster verification tool compared to software simulation based verification. Typically, emulation based verification tools come with a hardware accelerator card that helps to speed-up simulation and a software program that interfaces the card and the software simulation tools. The hardware accelerator card has one or more programmable devices and a set of fixed interfaces. The software takes the HDL code of DUT and partitions it to fit into the programmable devices on the card. Speed improvement in verification through emulation comes with an overhead of extra cost for the emulator hardware and design time. Hardware emulation platforms can cost up to a million dollars and can run only at a speed of 1 or 2 MHz This speed is 100 to 1000 times faster than simulation but still too slow for some applications like video processing. Considering the advantages and disadvantages, emulation based verification is a faster alternative compared to software simulation but it is very expensive. Also the speeds achieved do not give real-time performance and are much slower than expected for many applications. ASIC designs like video codec require faster and cost-effective verification techniques to reduce time to market and design cost [2] [3].

Field programmable gate array (FPGA) is a semiconductor device containing programmable logic blocks and programmable switches that interconnect the logic blocks. Also, FPGA are reconfigurable. These features of FPGA allow them to be used for any application and quick prototyping. ASIC designs are generally time consuming and are not cost effective for small designs and low volume production. FPGAs were introduced as an alternative to ASIC to shorten the time to market and overcome huge production cost of ASIC for small designs. As the gate density on the FPGA increased, they were quickly adopted into emulation based verification tools to significantly enhance the speed of simulation based verification techniques. FPGAs are also used to prototype a fully verified ASIC design for system level co-verification on custom designed boards before going for actual fabrication [3]. Many approaches and techniques for system level hardware software co-verification using FPGA have been suggested in [4] [5] [6].

B. Re usability Method

Verification reuse offers great opportunity to improve verification time. Reuse can be done in three manners:

a. Verification Components Reuse
b. Verification Plan Reuse
c. Verification Environment Reuse

The goal of verification reuse is to take advantage of existing modules belonging to a completed verification environment in other projects and subsequent generations of the same project. As such, the most fundamental requirement & the limitation for effectively reusing an object is that the part of the design that is being targeted by that object is not changed across different designs or multiple generations of the same design [3].

Verification components are in effect mini verification environments. Each component is targeted at a specific protocol,

interface or processor. Their primary mission is to reduce the effort required to construct and validate a verification environment. To be effective, verification components must be constructed as reusable, configurable environments which are packaged to plug-and-play [3].

Verification components are inherently reusable since they are encapsulated and are typically targeted at a standard specification. They can be reused when moving from module-level to chip-level to system-level verification as well as when moving from project to project.

Reusable components, such as patterns, BFMs, drivers, monitors, system service and scripts, can be reused either in different IP verification platform or from IP standalone platform to SoC verification platform. The design methodology is critical to the reuse of these components [7].

C. Abstraction-Based Method

While today's RTL design and verification flows are a step up from the gate-level flows of two decades ago, RTL flows are straining to meet the demands of most product teams. When designs are sourced and verified at the register transfer level, IP reuse is difficult, functional verification is lengthy and cumbersome, and architectural decisions cannot be confirmed prior to RTL verification. So here idea is to move to next level of abstraction above RTL to get a much-needed boost in design productivity.

That next level of abstraction is based on transaction level modeling (TLM). By creating TLM IP as their golden source, design teams can ease IP creation and reuse, spend less time and effort in functional verification, and introduce fewer bugs. Design iterations are reduced because TLM verification is much faster than RTL verification, and architectural choices can be verified well before RTL verification [8] [9]. Further, transaction-level models can be used for hardware/software co-verification, and can be part of a virtual platform for early software development. The net result of all these advantages will be much higher designer productivity.

Transaction-level modeling uses function calls, rather than signals or wires, for inter-module communications [9]. It lets users analyze transactions such as reads or writes rather than worrying about the underlying logic implementation and timing.

At the register-transfer level, the structure of finite state machines is fully described. This means that one needs to commit to micro-architectural details when writing RTL, such as the memory structures, pipelines, control states, or ALUs used in the resulting implementation. This requirement results in a longer and less reusable design and verification flow. The only way here is to move to an abstraction higher than RTL, which makes faster IP creation and design reuse. Moreover, by coding at a higher level, TLM IP requires fewer lines of code [9], and thus has fewer bugs. Functional bugs are detected and resolved earlier in the design cycle. The total verification effort can thus be greatly reduced.

D. Assertion-Based Method

System Verilog have some features to specify assertions of a system. Assertions specify a behavior of the design or system. Assertions are primarily used to validate the behavior of a design. In further, they can be used for providing generate input stimulus for validation and functional coverage. Today, assertion-based verification (ABV) has been applied at multiple stages(levels) of design and verification abstraction ranging from high level assertions within transaction-level test benches down to implementation level assertions synthesized into emulation and hardware[10][12].

The assessment of the assertions is guaranteed to be same between simulations, which is event-based, and formal verification, which is cycle-based. System Verilog allows assertions to communicate information to the test bench and allows the test bench to react to the status of assertions without requiring a separate API [10].

Assertion based Verification can also be used to achieve the goal of reduction in verification cycle time. Assertions are the

internal test points that wait for a certain predefined condition to arise and then notify the designer about the occurrence of the same. One of the major merit of assertions is that they become a part of the design. Modern assertion methods enable you to specify assertions and monitors in line with the HDL code for the module. Placing assertions in-line is main point as inline assertion cannot get lost [9].

Assertions can be utilized in various ways. They can be included directly within the hardware-description language (HDL) code that comprises the register-transfer level (RTL) description of the design or, they can be applied from outside in the form of test benches, or collections of test vectors, to check the response of the design to stimulus, or to control a stimulus generator or model checker. Assertions are properties or facts describing the required and forbidden behavior of a design. They are "executable specifications" that are monitored during simulation by assertion checkers included in the design file [11].

Assertion-based verification benefits users by simplifying the diagnosis and detecting bugs by localizing the occurrence of a suspected bug. It thereby reduces simulation-debug time significantly. Secondly, Self-checking code helps a lot in reuse of design and Interface assertions help find the interface issues early on [14].

E. Co-Verification-Based Method

Hardware/software co-verification is one of the techniques that can be used to begin the debugging process sooner, before physical prototypes are available. Today, most designers perform this only after the hardware has been developed. According to earlier studies, the sooner a bug is found in the design stage, the less expensive it is to fix [15].

This is especially true for SoC and ASIC designs where the high level of integration makes much of the design inaccessible to traditional prototype debugging tools. While there are debug facilities available for these designs, they are limited to the kind of data that can be gathered. On-chip debug buffers are limited to the amount of data that can be gathered. Some of these debugging facilities can change the real-time aspects of the system [15].

This leads to the potential of finding bugs that only manifest themselves when the debugging monitors are turned off. These are some of the hardest bugs to find. Using co-verification, you should have complete control over the system being debugged, as well as complete visibility into the operation of the hardware and software. An additional benefit is that a software model of the system will not contain any manufacturing defects. Often when checking out your code on a physical prototype, you will spend some time debugging hardware and manufacturing problems as well as debugging your code [15].

F. Model-Based Methods.

The last approach to analyze verification time and perform system verification earlier in the design process is to use model based design. Model based design is a method that emulates system [16] behavior using modeling and simulation. In other words, a virtual abstraction level is created. This abstraction level provides valuable insight into the hardware and software design.

In model-based design, a libraries of design models at the component and system levels are built. Then after we simulate these models to enhance system behavior, to analyze their designs, and automatically generate code to embed in deployed systems, apart from these models are also re-utilized for hardware-in-the-loop and other testing approaches.

While following the traditional design flow, a larger portion of the simulation and debugging work tends to occur later in the design, during HDL coding. With model based design the model defines the code, and therefore you are bonded to include in the model every detail needed to define the waveform.

Typically the models are built and tested accordingly, and deal with the bugs and the algorithmic issues as they occur. Debugging is handled entirely during the modeling phase of

the design, with a bit-true, cycle-true model [17].

By enabling earlier verification, Model-Based Design is helpful in teams to find defects, validate requirements, and confirm that design strategies are on track, and still there is time to address any problems that are discovered. Design defects and other issues discovered later in which development process are expensive to fix. When found early, these same problems can often be resolved with minimal impact on the schedule and the budget in Model-based design approach.

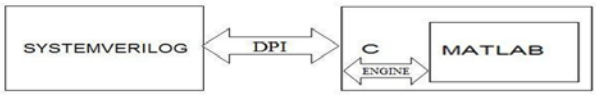

Fig 1. Co-simulation between matlab, C & System Verilog

By combining the above two Co-Simulation process, the System Verilog-C Language-MATLAB Co-Simulation was achieved. With this step we combine the power of System Verilog and MATLAB. To be more specific here we have combined the System Verilog DPI and MATLAB Application Programming Interface, with C Language common between the two bridges. This implementation creates a wrapper of C around MATLAB Engine and uses the DPI concept to communicate with SV as shown in the figure above.

Here the simulator tool will execute the SV code. When the DPI call to C Language based import function is done then the resident GCC compiler with Linux operating system is called for the execution of the C code during run time execution. Later the C code will have the Engine related functions and so the MATLAB Engine Library will be called. The final control of the simulation however remains with the System Verilog simulator. This was possible according to the command line linking library switches which were provided for simulation. The following command at the Linux terminal will launch the simulation.

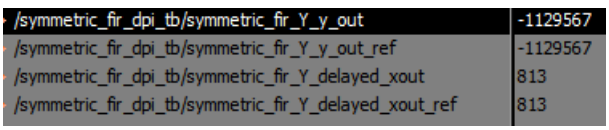

Fig 2. Simulation Result

As in above figure we can see the simulation result of symmetric Fir filter, in which Y_out_ref is generated using matlab and Y_out is generated using Model Sim. And both results are same. Here we have used DUT which is written in System C and test bench using system Verilog. And for co-simulation between C and System Verilog, we have used DPI-C method.

III SELECTION OF METHOD

Hence from the below difference and the tools availability the combination of Reusability, Assertion based method and model based method is chosen for the project. For Reusability I have used UVM methodology, System Verilog for assertion, MATLAB and System C for model based approach, c for co-simulation of MATLAB and System Verilog.

A. Verification Estimates

Methods		Verification time improvement	Level of Verification
Hardware based Methods	FPGA Prototyping	25.00%	****
	Emulation based	20-25%	****
Abstraction- Based Method(TLM)		17.00%	Up to RTL
Assertion Based Method		50.00%	Only RTL
Reusability		15-20 %	At RTL
Co-Verification Method		Medium	At Each Level (Application Base)
Model Based Methods		Very high	Any Level

CONCLUSION

By Different estimates, verification constitutes up to 60-70% of a typical design verification time and budget. The described methods help to effectively decrease the verification time. However each method has its own limitations and demerits. By the results, it

justifies that the best technique to reduce verification time effort is model based method. We believe that it would be a very useful technique for reducing verification timings.

REFERENCE

[1].Zeineb Bel Hadj Amor, Laurence Pierre, Dominique Borrionr, "System-on-Chip Verification: TLM-to-RTL Assertions Transformation", 14516132, June 30 2014-July 3 2014

[2].Basavaraj Mudigoudar," FPGA Prototyping for fast and efficient verification of ASIC H.264 decoder", The University of Texas at Arlington, May 2006.

[3].Nirav R. Parmar, Vrushank M. Shah "Study of Various Ways to Optimize ASIC Design Cycle timings", ISBN: 978-960-474-155-7

[4].S. Brown and J. Rose, "FPGA and CPLD architectures: a tutorial", IEEE Design & Test of Computers, vol. 13, issue 2, pp. 42 – 57, Summer 1996.

[5].P.-A. Hsiung, "Hardware-software timing co-verification of concurrent embedded real-time systems", IEE Proceedings on Computers and Digital Techniques, vol. 147, issue 2, pp. 83 – 92, Mar 2000.

[6].Y.Sungjoo et al., "Fast hardware-software co-verification by optimistic execution of real processor", Proceedings of IEEE Conference and Exhibition on Design, Automation and Test, pp. 663 – 668, Mar 2000.

[7].Rui Wang , "Reuse issues in SoC verification platform", Pages: 685 - 688 Vol.2 , ISBN : 0-7803-7941-1 , 26-28 May 2004

[8].http://www.soccentral.com/PrintPage.asp?PassedEntryID=18241

[9].Steve Brown, "TLM Driven Design and Verification", white paper, www.cadence.com,June 2009.

[10].http://www.testbench.in/AS_00_INDEX.html

[11].ZocaloTech,www.zocalotech.com/files/assertionverification_whitepaper.pdf

[12].https://verificationacademy.com/courses/assertion-based-verification

[13].Yangyang Li , Wuchen Wu , Ligang Hou , "A Study On the Assertion-Based Verification of Digital IC" , 2009 Second International Conference on Information and Computing Science

[14].Mcmillan, K.: Assertion-Based Verification, Cadence Berkeley Labs (2005)

[15].Russell Klein "Solving problems early on using
Co-verification", SoC Verification Business Unit
Mentor Graphics Corp.,Nov 2004.N

[16]. The Power of Modeling, www.20sim.com

[17]. Advantages of Model-based design, www.xilinx.com

[18].http://www.design-reuse.com/articles/16358/a-new-methodology-for-hardware-software-co-verification.html

[19].http://in.mathworks.com/solutions/verification-validation/