IMPROVING VERIFICATION METHODOLOGIES IN DIGITAL CIRCUITS MODELING

Iulian NIȚĂ[1], Adrian RAPAN[2]

Evoluția tehnologică aduce noi avantaje prin îmbunătățirea metodelor de verificare pentru proiectarea circuitelor digitale. Ca un prim pas, standardul industrial Accellera Universal Verification Methodology (UVM) 1.0 a fost lansat de curând, pentru a reduce limitările existente în MDV (metric-driven verification) şi pentru a ajuta inginerii în accelerarea procesului de verificare funcțională. În acest articol sunt prezentate cele mai importante metodologii de verificare, sunt subliniate provocările existente şi sunt trasate principalele direcții ce trebuie urmate pentru obținerea unor implementări hardware mai bune. Folosindu-ne de aceste metode, am propus o nouă arhitectură pentru predictor, care, în comparație cu metoda clasică, aduce îmbunătățirea procesului de depanare, creşte rata de detecție a erorilor şi acoperă mai multe scenarii. Rezultatele au fost validate prin verificarea comparativă între cele doua metode pe un bloc funcțional UART scris în Verilog.

Significant new advancements are announced day by day to help boost the verification productivity for system on chip design teams. As a first fresh step, the emerging Accellera Universal Verification Methodology (UVM) 1.0 industry standard is here to expand the limits of metric-driven verification (MDV) and help engineers to achieve faster and more comprehensive verification closure and fast tapeout. This article presents some of the key elements of the verification methodologies and flag the market needs for better silicon realization on the context of growing electronics innovation. On top of recent methodology releases, we propose a new predictor architecture that, compared to classical approach, brings debug process improvements, increased bug rate detection, meaningful functional coverage. This approach targets all points of interest in digital design verification. As a practical example we have compared a legacy verification environment of an opencore UART IP Core (16550) written in SystemVerilog VMM.

Keywords: ASIC verification, UVM, OVM, TLM

1. Introduction

As design complexity is growing day by day, the verification methodologies are continuously updated and the verification flows become fractured and, in some cases, inefficient. Though each technology is presented with a faster bug detection rate than previous, the fact is that engineers leveraging

[1] Assist., Depart. of Applied Electronics and Information Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: iulian.florin@gmail.com
[2] Lead Verification Engineer, Cadence Design Systems, Edinburgh, UK

experience on top of abstraction capabilities of each of the methodology is making the true difference for Silicon realization.

Massive functional capacity, performance and strict power management, are embedded in recent SoC realizations. The verification challenges on such complexity scale are growing, and traditional verification methodologies are losing field as verification costs must be met, and the time to market must be reached. Another burning point on today's electronic innovation is the exploding software content, and what is needed is a methodology that allows both easier software development and faster silicon realization. In the past, it was possible to have a early silicon sample for software development.

ITRS (International Technology Roadmap for Semiconductors) provided several projections for the future. On a recent report, we can clearly observe (Figure 1) the increasing HW-SW design gap.

Such methodology that enables a unified software and hardware development has as a starting point the specification analysis of the whole system for each of the three directions: design, verification, software development.

In such a unified development environment, the verification role has increased, from developing classical test benches, to complete architecture of transaction-level models that enable architecture testing, performance metrics, software development, and accurate and efficient design verification. We can now start seeing, how the transaction-level verification improves the productivity gain, and silicon realization flow.

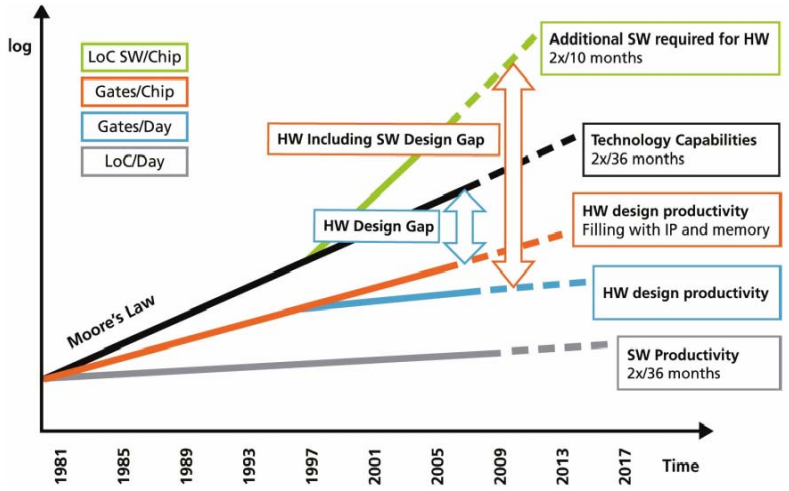

Fig. 1. HW – SW Design Gap: ITRS report [1]

On top of all methodology progress that is driving the industry to efficiency improvements, we must also take in consideration the experience

capabilities and the guidelines that are followed when building such a complex verification environment. The user guide of each methodology is the main point of interest when an engineer starts it's project work, but the examples are simple and must not be followed in complex verification environments architecture. We propose some practical enhancements: an improved predictor architecture that compared to classical approach brings improvements to key elements of the verification process: debug, bug rate detection, and functional coverage.

2. Challenges

Within a motivating economy, and selective consumer, semiconductors companies must satisfy the demands of increasing application capabilities, energetically efficiency, and more important, a quality bug free silicon.

As shorter project schedules appear, and rising complexity is beside, the engineering challenge becomes without limits if certain methodology is not followed, and improved by the personal experience and abstraction capabilities.

On top of this global complexity challenge, there is also a lack of interoperability of different tools used, and reuse factor from different projects.

The challenge in the current context is resumed to the fact that designs are developed from the perspective of user applications demand, with certain functional characteristics, meaning that this is a significant change from the era when semiconductors releases drove the design progress.

3. Verification principles

Functional verification of a design is practically resumed to comparing the designer's intention with regards to a certain functional behavior, in order to determine equivalence between the two thinking paths. Verification is the form of checking that the design is suitable for production. The verification engineer's effort is driven by the specification, resulting in a verification plan. The simulations will exercise the design, having certain random or directed scenarios. Today's tools & methodologies assure us the needed automation in terms of pseudo-random stimuli generation, coverage collection, and verification environment architecture. Coverage collection helps us to identify the areas that need more attention, or holes that must be filled with particular test cases.

4. Moving to a new level of abstraction

UVM, (Universal Verification Methodology) is a standard released in May 2010 by Accellera with the purpose of unifying verification interoperability, targeting productivity increase. The background of this standard in based on the several solutions that appeared along time to address the growing verification

challenge. Can be mentioned: Cadence eRM (e Reuse Methodology), SystemVerilog VMM (Verification Methodology Manual), OVM (Open Verification Methodology). Each one had its own good parts, but this separation leaded to a strong divergence in the verification world. The release of UVM 1.0 EA showed the true industry collaboration can lead to remarkable technology improvements in the verification ecosystem.

Currently, one of the most targeted standardization trends is the use of transaction-level models for architecture building, design and verification. Transaction-level modeling enables interoperability between virtual prototyping, integration in RTL design flow, testbench acceleration by TBA(Transaction Based Acceleration), and reuse in verification environment. As shown in Figure 2 today's design flow involves design at several levels of abstraction.

Transaction-level modeling is the one of the principles that has driven the release of OVM (Open Verification Methodology) followed by unifying UVM (Universal Verification Methodology), enabling the full support for verification environment creation to be applied to TLM, RTL, and RTL acceleration refinement (a technology that enables simulation acceleration using complex hardware (Cadence's Incisive Palladium XP) for real time stimulus driving for computation speedup). On such machines, UVM is supported, and this brings an massive acceleration of logic simulation just by using transactions as communication granularity between components running on the workstation.

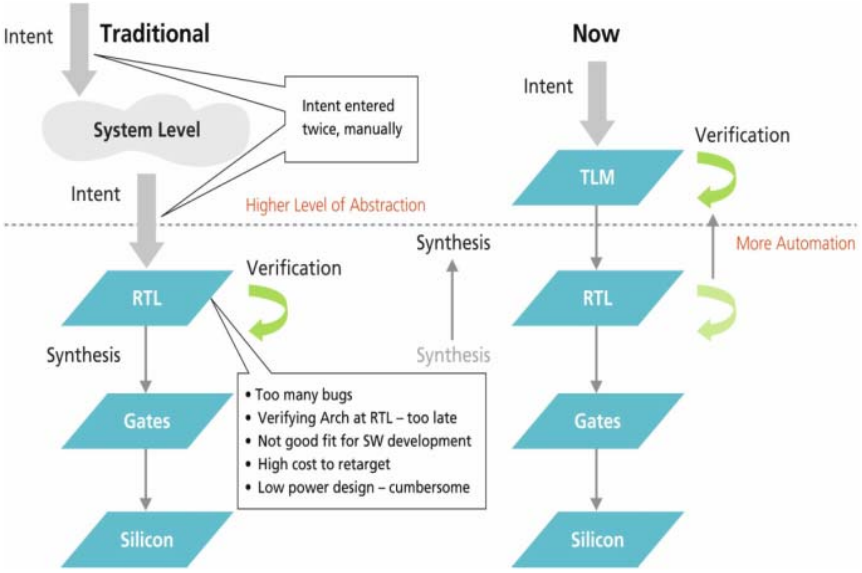

Fig. 2. RTL flow vs. TLM based flow [1]

Add UVM reuse, constrained random stimulus, functional coverage & checking, and you have a system level productivity improvement

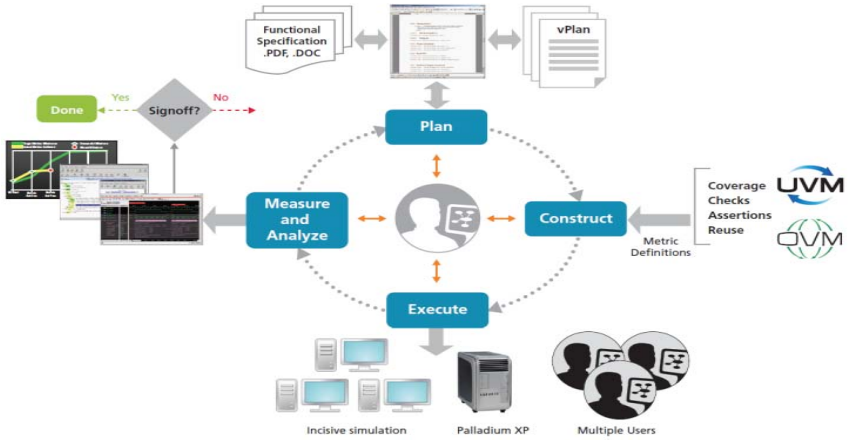

Fig. 3. Metric-driven verification flow [2]

5. Verification environment improvements

Though each technology is presented with increased capabilities than previous, the fact is that engineers leveraging experience on top of abstraction capabilities of each of the methodology is making the true difference for Silicon realization and faster bug detection rate.

When building a verification environment, the engineer has in mind typical verification environment architecture, influenced by the simple examples from methodology user guide. The skeleton of a VE was roughly the same from project to project when the complexity was at acceptable levels, but we must take also in consideration how fast a VE finds the first bug in blocks with complex functionality.

The classical (Figure 4) approach was that the predictor or the reference model at a higher level of abstraction is fed by the transactions decoded on the inputs, and in an infinite loop the predictor detects the functional state and issues appropriate response emulation and selects certain verification actions. With this approach there is no link between the possible functional states, the selection having always the same starting point (“Detect state“Figure4)

This classical approach downside has been proven to be a poor first bug detection convergence and the effort during debug (which is approximate 60% from the development time of a verification environment) was significant.

Coverage collection is also deficient in terms of transition coverage between functional states.

The proposed predictor architecture (Figure 5) comes to overcome the downside of the classical approach.

This predictor architecture approach has a starting point the maximization of functionality segmentation, and task splitting. Every action of the predictor must be clear separated, and each functional state must be correlated with other. The architecture is similar to building a finite state machine. Each state has its own trigger event. When a specific state is reached actions (emulation, checking) are issued and then a jump to another state is made. This approach comes very good when used in TLM development, where we have a transaction modified several times considering previous actions of each particular state.

Improvements may be added to this architecture, as we do not propose a golden ticket to fast bug rate detection. The approach may vary from project to project, but keeping the base idea of partitioning and linking complex functionality in order to touch each aspect of a complex block, will definitely hit all mentioned improvements.

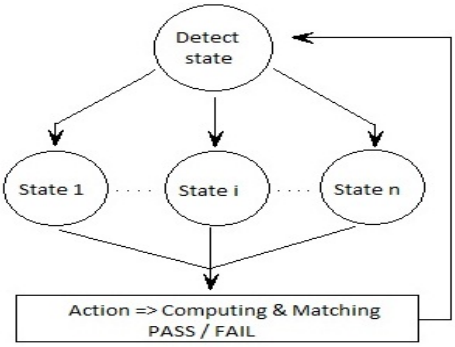

Fig. 4. Classical predictor architecture

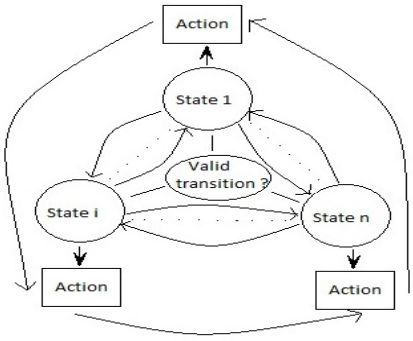

Fig. 5. Improved predictor architecture

For our experiments we used Specman [6], which is an EDA tool. It provides advanced automated Functional verification of hardware designs and an environment for working with, compiling and debugging testbench environments written in the *e* [7]. (**e** is a hardware verification language (HVL) which is tailored to implementing highly flexible and reusable verification testbenches). Specman also offers automated testbench generation to boost productivity in the context of block, chip and system verification (figure 6). Using "e" language flexibility with regards to events, temporal expressions, aspect orient programming, and much more, the implementation was robust and flexible.

As a practical example we have compared a legacy verification environment of an opencore UART IP Core (16550) written in SystemVerilog VMM. The legacy solution was based on the classical architecture building approach. We have as a comparison base the schedule and the project plan and estimates at that point. We implemented a second verification environment using UVM and proposed predictor architecture.

The comparison resulted in:

1. Debug improvements were massive: separating functionalities allowed us to out the RTL to work as the developing of the environment was progressing. Once a state corresponding to a particular functionality we intent to verify is coded, this enables us to create stimulus to hit that area and debug any problems if any detected.
2. Having the granularity as distinct states meaning replicating certain actions of the predictor, made the identification of root cause of test failing to be fast and accurate.
3. Bug detection rate was improved: we have hit the first bug after 3 hours, comparing with the 15 hours of the legacy verification environment as from the plan and bug reporting database.
4. Functional coverage was written for each state and also for transitions between states: we had the proof that our verification environment has hit all our intentions from the plan.

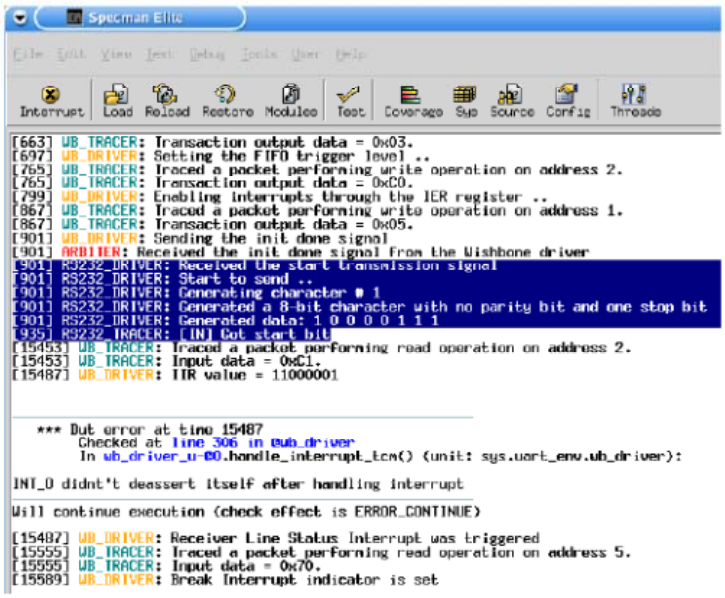

Fig. 6. Snapshop from Specman (error detection)

The advance of technology in semiconductors industry pushes very often a lot of structural changes to legacy modules. If such module was verified using a predictor built following proposed guidelines, the changes needed to align the verification environment to specific requirements can be very fast estimated and implemented.

6. Conclusions

Transaction-level modelling enables architecture concept testing and refining, early software development, good starting point for RTL development, full reuse of the TLM in verification environment. This approach is needed to enable the modern technology system design and verification complexity.

Custom verification environment components may lead to significant improvements on debug time and improved bug rate detection as the proposed predictor architecture had on the example comparison.

The predictor (reference model of the device under test) is the most important component of the verification environment. Most of the debug time is spent around this area and if a specific architecture or guidelines come to improve this time, it should definitely be taken in consideration during development. Our proposed architecture offers improved debug time, a robust structure that is easy to maintain for future changes.

REFERENCES

[1] TLM-Driven Design and Verification Methodology, Cadence Design Systems : http://www.cadence.com/misc_pages/publications/tlm/index.aspx

[2] Comprehensive UVM/OVM Acceleration – White Paper –Cadence Design Systems: http://www.cadence.com/rl/Resources/white_papers/UVM_OVM_Accell_WP.pdf

[3] *Kasuya, A., Tesfaye, T.*, "Verification Methodologies in a TLM-to-RTL Design Flow", JEDA Technol. Inc., Los Altos ; Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4261171

[4] *Bruce, A. Nightingale, A. Romdhane, N. Hashmi, M.M.K. Beavis, S. Lennard, C.*, "Maintaining Consistency Between SystemC and RTL System Designs", 2006 DAC

[5] *Doug Smith and David Long*,"Stick a fork in It: Applications for SystemVerilog Dynamic Processes", SNUG2010

[6] Specman Tutorial, http://www.asic-world.com/specman/tutorial.html

[7] Specman verification, http://www.specman-verification.com/