sponsored by

Gabe on EDA • EDAmarket

Assembling the Future

A Newsletter About the Design and Production of Electronics

ISSUE 015 • MAY 2012

SUBSCRIBE

PDF and Archives

twitter

In this issue:

Virtual Prototyping and More Discussion on Standards

Gabe Moretti

The editorial calendar for this month listed Virtual Prototyping as the subject. I was surprised not to receive an article from Synopsys, the market leader in this segment. But the mystery is likely to be solved at DAC, according to rumors from well informed sources. So yet another reason to go to DAC and see what the largest EDA company is up to.

This issue contains one article from EVE about virtual prototyping. The author is Ralph Zak, an EDA veteran with a strong background in hardware prototyping. Ralph has recently joined EVE as Business Development Director. Ralph proposes an hybrid approach to system level development that uses both virtual models at the system level and RTL code executing on an emulator. The only reasonable alternative to this approach is to build a prototype in hardware that contains both the cores and the reused IP that would have to be implemented in FPGAs.

The core message of the article is that SoC design teams rely on virtual prototyping early in a project, and then transition to the use of hybrid transaction-based virtual platform-SoC emulation systems for fast, pre-silicon software development and debug. This hybrid environment provides high-performance debugging for software using an IDE with virtual processor models and RTL debug proceeds using the emulator debug features.

By the way for a good discussion on prototyping you may want to attend the DAC pavilion session on Wednesday at 4:30 PM. The panelists will discuss the merits of building a specific prototype versus using a commercially available one.

The other two rticles in this issue look at standards from a pragmatic point of view.

Michiel Ligthart of Verific writes about a problem I encountered in my previous profession, as a VP of engineering at VeriBest. Standards are supposed to allow the industry to have portability and interoperability through a well defined and understood set of rules and definitions. For modeling languages these are defined in the syntax and semantics of the language. Yet, Verific, just like VeriBest, found that there are different interpretations of a language standard. Some get clarified in the next revision of the standard, and some live on to provide subjects of debate among language architects.

Michiel has titled his article: "Standards maybe silver, but in EDA implementations are golden". It says it all in one brief sentence. But, just like language standards the devil is in the detail, and reading the article you will find the details.

Lauro Rizzatti of EVE writes about the same problem but with a business point of view. The reason that there are different implementations of the same standard is not just because there can be different interpretations of a written standard, but because, observes Lauro, companies have existing customers to protect. Companies invest a significant amount of resources in establishing both a methodology and a library of IP when using a particular tool. When a new standard impacts the way a tool works, implementing it may in fact require existing customers to make changes. Change is expensive both in what needs to be done and in the opportunity for errors. The best case I specifically witnessed was the Mentor implementation of the 1987 IEEE -1076 standard: the VHDL language. Mentor, who had purchased Model technology a pioneer implementer of VHDL before the language was standardized had made a semantic choice that was not literally incorporated in the standard. So Mentor's VHDL, which was the simulator with the largest number of users, did not in fact follow the standard.

At VeriBest we implemented the correct standard but put in "The Mentor Switch" to allow VHDL code written for the Mentor simulator to work exactly as it would on the original simulator when users switched from Mentor to us.

ESL Virtual Platforms and Emulation for Early Software Development

Ralph Zak, Business Development Director, EVE

As SoCs double in number of processors and incorporate twice the software content with each product generation, software development moves to the forefront of concerns for development teams and project managers. ESL Virtual Platforms provide high-level models of new SoC designs, including processor models. These instruction-accurate models run fast and provide a suitable platform for early software development. As the design evolves, overall accuracy improves with cycle-accurate RTL code for major design blocks. Co-simulating the ESL virtual platform with available RTL code in simulators reduces the performance, and such a hybrid environment is simply not suitable for continued software development.

However, a hybrid environment of ESL virtual platform with a cycle-accurate, hardware-based SoC emulation system maintains the high-level performance needed by software development teams. As the full design implementation model becomes available, the hybrid ESL-emulator environment gives way to the emulation system as the validation transitions for both software and hardware teams. This development flow from ESL to hybrid ESL-emulation to emulation-based validation provides a continuous high-performance environment that increases in accuracy over the project cycle. Providing early and increasingly accurate models for software development throughout the project minimizes the risk of software extending the project critical path and causing unacceptable delays in project schedules.

ESL Virtual Platforms for Early SoC Optimization

ESL systems are available from a number of EDA companies, including Synopsys and Carbon Design Systems. Virtual Prototyping systems provide SoC modeling technologies in C, C++ and SystemC. When a high-level model is generated, it becomes an executable specification of the design. While not cycle accurate, it represents the full intention of the design team.

Typically, the high-level model is used to optimize the design and partition it into its hardware and software elements. As processors are selected for the design implementation, the ESL virtual model is updated with instruction-accurate models of the processors.

In a typical design, the evolved virtual platform can be leveraged as a software development platform using the instruction-accurate processor model with appropriate software integrated development environments (IDEs). Execution speeds are normally in the tens of MHz or more for these high-level models. A drawback to such environments is that they are not cycle-accurate and, therefore, timing-dependent faults are not discovered. Eventually, the software must be validated in a cycle-accurate environment, long before real silicon is available.

Emulation as a Complement to ESL Virtual Platforms

Design reuse has grown to 50-80% of many SoC designs, which means most of an SoC design's RTL code is available in the early stages of a project. Consequently, much of the design can be instantiated on hardware-based SoC emulation systems during the early stages of a project. This increasing availability of "implementation" level parts of the design provides design teams with

several options:

1. The software can continue to be developed on the SoC's high-level model with specific processor models

2. The available RTL from design reuse and licensed IP suppliers can be integrated in a co-simulation environment consisting of the ESL virtual model and the design team's RTL simulator of choice

3. A co-emulation environment can be set up between the virtual platform and an emulation system with the available RTL code implemented in the emulation system and run at hardware speeds

If the design team elects to go with the first option, they see no improvement in the accuracy of the software development environment. Any verification issue related to timing will not be discovered until later in the project schedule, and increase the attendant project risk.

Going with mixed-level co-simulation will result in a slow environment, typically running at tens of Hz, too slow to effect any real-time verification. As a result, considerable frustration will result as functions like booting an operating system would take weeks, months or years, instead of minutes.

With the third option, the hybrid ESL-emulation system will typically run at MHz speeds, sometimes 10 MHz or more depending on the design, allowing for most software to be executed and debugged. Such an ESL co-emulation environment, with the ESL virtual prototype and the available RTL code implemented in a fast emulation system, will allow software debugging to proceed with increasing accuracy as more and more of the design is made available as RTL code.

The high-performance co-emulation integration between the ESL C, C++, SystemC world and RTL emulation can be implemented using a transaction-based communication layer of synthesizable verification IP, minimizing the traffic between the ESL Virtual Platform and the cycle-accurate emulator.

For example, EVE provides C/C++ APIs for its ZeBu emulation system and all of its application-specific synthesizable verification IP, as well as pre-integrated interfaces to specific commercial ESL systems to implement such hybrid environments.

Emulation for Software Development

To make hybrid virtual prototype-emulation system software development environments accessible to every design team, the emulation systems must be scalable to emulate the SoC of a specific project, from a few million gates to billions of gates. Also critical to maintaining the integrity of the overall software development effort is the performance of the hybrid environment.

By the very nature of an emulation implementation of a design across multiple FPGAs, custom emulation SoCs or custom emulation FPGAs, the larger the overall system and the more chips it takes to implement a design, the slower the environment. Size does matter in a hybrid environment. Smaller, denser systems run faster. If used early in a project, these systems must have well-established RTL debug capabilities and established methodologies for tracing RTL design problems

and supporting software development. Project teams frequently attempt to provide low-cost hardware platforms or prototypes for larger software teams while hardware teams debug on the emulation system. Difficulties usually arise from having to re-target the design to a lower cost platform, which can take months.

If the desire is to transition from pure virtual prototype to hybrid to emulation as RTL code becomes available, re-targeting can become quite painful. The ideal environment is one where the same design implementation is used for both the hardware emulation debug environment and the lower-cost software emulation platform. An example of how this can be implemented is with EVE's ZeBu, available in Hardware Development Platform (HDP) and Software Development Platform (SDP) configurations. Both systems represent the SoC identically and use the same programming stream to configure them.

Consequently, once the hardware team has mapped the SoC design into ZeBu, whether specific blocks or the entire SoC, multiple SDP replicates can be programmed using the HDP configuration data. The software team does not need to know how to configure ZeBu, or anything about the SoC RTL code.

Bypassing Virtual Prototypes to Develop Software

If the processor model is supplied as RTL code, the entire design can be implemented in the emulation system. It needs to support on-going software development in this environment with flexible methodologies to accommodate the needs of different project teams. Some emulation systems support connecting the software IDE through a number of mechanisms to provide flexibility, such as:

1. JTAG transactors to bridge the host-based IDE directly to JTAG ports on processors being emulated
2. For the Cortex family of ARM processors, ARM provides a VSTREAM transactor to link its own IDE to high-speed serial debug ports of the processors
3. Connecting traditional hardware JTAG ICE systems to emulated embedded processor JTAG ports

Summary

SoC design teams rely on virtual prototyping early in a project, and then transition to the use of hybrid transaction-based virtual platform-SoC emulation systems for fast, pre-silicon software development and debug. This hybrid environment provides high-performance debugging for software using an IDE with virtual processor models and RTL debug proceeds using the emulator debug features. Ideally, the software team uses lower-cost hardware systems in their hybrid environments that do not need the SoC RTL code to be re-targeted to the software development platform. Once processor RTL code becomes available, the emulation link to the virtual prototype can be severed, and debug of hardware and software can be done on the full implementation emulation model of the design.

About Ralph Zak

Ralph Zak is director of Business Development at EVE. He has held senior management positions in many design automation companies over the past 30 years. His responsibilities have included strategy development, product planning, technology partnerships and marketing. Several of these companies prior to EVE also offered hardware-based verification systems, including HHB Systems, Quickturn, Aptix, GiDEL, and Mentor Graphics. Zak has a Bachelor of Science degree in Mechanical Engineering from the University of California, Berkeley, and an MBA from Stanford University.

Standards maybe silver, but in EDA implementations are golden

Michiel Ligthart, President and Chief Operating Officer, Verific Design Automation

Verific Design Automation has made it its business to provide VHDL, Verilog, and SystemVerilog parsers based on their respective IEEE standards to the EDA community over the years. This has not always been an easy task, but it has kept us in business and we obtained the respect from our peers, our customers, and the end-users alike.

The IEEE standards for VHDL and SystemVerilog (of which Verilog is a subset) have drastically evolved over time, most evident by their sheer sizes. The first VHDL standard, IEEE 1076-1987, is a relatively small book of 218 pages, cover to cover. (I still have my hard copy.) Over time the standard was upgraded, first to -1993, then -2002, and today we are at the -2008 version. And just like the American populace, the standard gained significant weight during that period as it now sits at a bulky 640 pages.

That may sound like a lot, but it is still modest compared with what SystemVerilog did to us. The first one, IEEE 1800-2005, is only 664 pages. But wait, there is more. These are in addition to the IEEE 1364-2005, regular Verilog, which adds 590 pages of its own. Because there is a lot of overlap between 1364 and 1800, the committee overseeing these efforts decided to merge them in a single document of 1285 pages, known as the IEEE 1800-2009. (An updated version soon to be known as -2012 is circulating in draft form and added only 17 pages.)

One problem we occasionally encounter is that of incompatible implementations of the same standards. In hardware, such a thing would be incomprehensible, but software implementations seem to get away with it. Just imagine if CISCO routers would implement a slightly off version of the wireless IEEE 802.11 a/b/g standard. Unless everybody uses a receiver from CISCO with the identical flaw, reception of wireless data from this device is likely to fail. Because this is not good practice, providers of such equipment make sure they meet the standard.

Not so though in EDA. At Verific, we (at least try to) adhere to the VHDL and SystemVerilog

standards to the dot. If the language reference manual (LRM) specifies that a construct requires a label, we check that that label is there and throw a fit if it isn't.
Not everyone does that. As our parsers became more and more proliferate in VHDL and SystemVerilog land, we found ourselves in situations where we would issue error messages only to be told by a designer that some legacy tool XYZ, on which they had developed their SystemVerilog, parsed it just fine. And even when the user would agree that his or her SystemVerilog contained a blatant violation of the LRM, their design was beyond the point this could be remedied. Our software had little choice than to downgrade the error message to a warning and let it slide.

These downgrades cannot be made permanent because, if that same designer would have used a different simulator, they would have complained that our parser did not catch this obvious violation of the LRM. Over the years, we have collected many of these discrepancies and compounded them under a "Relaxed Checking Mode" switch. Our software always had the ability to downgrade error messages on the fly so users in the field could continue if they encountered problems, but the switch has the added benefit that these specific downgrades are sanctioned by our parsers. A problem of course is that those situations where we have to downgrade our error checks are found only by trial and error. By definition they are not in the standard and never published.

The bottom line is, as we found out, the end-user of EDA tools cares more about EDA tools than about standards. And, if their RTL code is validated by one favorite EDA tool they like, they like their other EDA tools to adhere. At Verific, we concluded that it was easier to change our parsers and reproduce the behavior of that tool than change the RTL code of the designer. All we can hope for is that eventually they all will use our parsers to validate their SystemVerilog or VHDL.

Here is a final anecdote about adjusting our parser to reality (erroneous SystemVerilog that used to parse in some other EDA tool) instead of sticking to purity. It has happened on more than one occasion that a customer of ours requested us to implement something against the LRM. We consider this a customer specific enhancement and will comply if we cannot talk them out of it. In this particular case, and I am sure some of you see this one coming: within 12 months, the same customer filed a High Priority defect against its own enhancement. They could not believe we did not flag this error.

About Michiel Ligthart

Michiel Ligthart, Verific's president and chief operating officer, has an extensive background in engineering, product marketing and general management. Prior to joining Verific, Ligthart was vice president and general manager of west-coast operations for Theseus Logic, a startup in asynchronous logic. Before that, he spent eight years at Exemplar Logic working in engineering and marketing roles. Ligthart started his career with Philips Research Labs in California, and was a visiting scholar at the Center for Integrated Systems at Stanford University. He has a Master of Science degree in Electrical Engineering from Delft University of Technology, The Netherlands

---

The Case for Non-Standard Development

Lauro Rizzatti, General Manager, EVE-USA

There is no doubt that industry standards are a requirement for successful electronic design. At the product level, bus and protocol standards allow disparate components to be combined into integrated systems greater than the sum of their parts. At the tool level, EDA language, file format, and methodology standards enable reuse and portability, improve knowledge transfer, and encapsulate best practices across the industry. Yet despite these benefits, many EDA companies still continue to develop and promote their own, non-standard or proprietary solutions—even when their products are also standards compliant.

EDA standards typically first evolve out of fragmentation. Multiple companies take different approaches to solving the same problem—say, low-power optimization or property specification—and create a new language, format, and/or methodology as a part of their solution. As these individual solutions mature, design teams demand a consolidation of features and portability across tools, driving the need for a standard.

Hardware Verification Languages (HVLs) are a good example. Verisity (now Cadence) created e while Synopsys created OpenVera. After many years of competition, eventually SystemVerilog was developed, consolidating features from not just these HVLs, but also from various assertion and design languages. Offering reduced syntax, consistent behavior, and encapsulating state-of-the-art verification capabilities, SystemVerilog has since become ubiquitous in SoC verification. This is an idealized example of standards development. In reality, issues with standards adoption, application, and development all encourage parallel proprietary development.

When a new standard is released, not everyone immediately jumps on board. Although the Universal Verification Methodology was approved by the Accellera standards body in February 2011, many companies are still using the Open Verification Methodology (OVM) or Verification Methodology Manual (VMM) for their SoC verification environments. Some companies will delay the adoption of a new standard because they know that the tool vendors will need time to mature their product offerings. Many others choose to wait because of existing IP.

These companies have built up a library of IP and do not want to spend the time and effort to re-write and re-test previously verified code. For many companies, an interoperability strategy is a better choice—to keep using their existing IP in conjunction with new development using the standard.

Due to their existing customer base, at the very least, EDA vendors are required to maintain their current level of support for proprietary flows. Since there are likely to be new requests for bug fixes and for interoperability between the existing flow and the new standard, there must also be a code stream for new development in the non-standard flow moving forward.

A second issue driving proprietary development is the generalized nature of standards, and how this affects their application and performance. There isn't much point in an industry-wide standard if it cannot be applied to a large audience, but this application requires a certain level of generalization, and some features may fall through the cracks. If the missing feature is important enough, it may prevent a design team from moving to the standard.

In the emulation space, Standard Co-Emulation Modeling Interface (SCE-MI) is the Accellera standard that specifies the communication mechanism between the software and the hardware parts of synthesizable transactors. It is a low-level communication standard and does not specify how to write a transactor. Unfortunately, the specification may affect adversely the performance of transaction-based co-emulation, taking away one of emulation's key advantages. This is one reason why many design teams using EVE hardware-assisted verification platforms opt to continue with our proprietary solution, even though ZeBu supports SCE-MI.

The latest incarnation of SCE-MI, version 2.0, does include performance-based optimizations, but the implementation highlights another potential application issue—conflicting standards. The high-performance implementation in SCE-MI 2.0 includes a pipe interface not SystemVerilog compliant, which then reduces its practical usefulness. Sometimes a proprietary solution can actually be the more compatible one.

A final issue to consider is the amount of time it actually takes to develop and approve a standard. A standard can be seen as a unified snapshot of the industry at a given time; however, it can take months or years to develop this snapshot. UVM 1.0 took approximately 15 months to develop. In the meantime, the EDA industry is not static. While a standard is being developed and approved, EDA vendors must continue to develop their next-generation technologies, and it is easier for them to do so within their proprietary environments. This effort is still valuable to the standards process though, as these newer technologies contribute to the next version of the standard.

Non-standard or proprietary product development is a critical component of the EDA ecosystem that should complement, and not compete with, industry standards. This development maximizes reuse when new standards are being adopted, provides critical capabilities that may not exist in the generalized standard, and enables continued innovation that can be leveraged in future standards.