System Level Virtual Prototyping becomes a reality with OVP donation from Imperas.

Brian Bailey – EDA Consultant

Abstract

For many years, Electronic System Level (ESL) design and verification has been on the cusp of widespread adoption, but never seems to get there. Universities and companies claim to have the necessary breakthrough only to see the technology sit there for years or the company to hobble along without ever really becoming a success. Is this because ESL is failing to deliver on the promises or that the products are flawed? Is it because the preconceived notion of ESL is wrong? In this whitepaper the reasons behind this are investigated and the roles of a System Level Virtual Prototype (SLVP) will be discussed. Imperas Inc. recently announced their Open Virtual Platforms (OVP) initiative which may provide the missing piece of the puzzle to jump start successful commercial deployment of ESL flows.

Introduction

Electronic Design Automation (EDA) flows are built on the fundamental premise that models are freely interchangeable amongst vendors and have interoperability amongst them. This means that models can be written or obtained from anywhere and it is known that they will work together and be accepted by any vendor's tools for simulation, analysis, synthesis or for any other purpose supported by the model. While this seems like a simple goal, it has been completely elusive in the world of ESL. We have neither model interoperability nor independence between model and tool. Because of this, it is almost impossible to put together complete ESL flows today, and thus ESL adoption remains patchy at best, and often derided.

However, there are big changes happening in both the hardware and software worlds that will soon make it impossible to construct systems without an abstract model. In the hardware world, the large scale adoption of reuse means that larger systems are being put together like a Lego system. At the same time, processor complexity has hit a wall created by diminishing performance gains at the expense of huge power increases, such that most systems are now utilizing multiple simpler heterogeneous processors rather than one central processor. Much of the system functionality comes from software and this continues to grow as before, but now has to deal with the transition to a multi-processor world as well. These changes, on both sides of the system, mean that we cannot continue for much longer without a viable system level model on which this functionality and architecture can be designed and verified.

Historical Perspective

Hardware-software co-verification

For the past decade there have been attempts to bring the hardware and software communities together by providing virtual models of the hardware that could be used for software development and verification. Perhaps the most successful of them was Seamless[1] from Mentor Graphics which substituted Instruction Set Simulator (ISS) models for each of the processors and integrated them into a conventional RTL simulation environment. This provided a low level virtual prototype that was useful to the software developers since it provided them with something that looked very much like their familiar IDE and debugger and at the same time provided the hardware engineers with their traditional view of the system, including waveforms. This solution was found to be very good for driver debug and integration but did not have sufficient performance for much else. The Seamless product also included a number of performance boosters that virtualized the host memory system and this extended its usage into some low level OS areas[2]. Fundamentally though, Seamless is a cycle accurate RTL virtual prototype that is quickly bogged down by any activity within the hardware simulator. In later years faster models replaced the RTL models, such as C or SystemC models[3] moving it into the realm of cycle approximate execution. This provided a much needed performance boost but it still meant that complex systems operated too slowly, making it unsuitable for mainstream software usage. Other companies have tried similar strategies, but with similar limitations to performance and ultimately limited success. While this bottom up product development strategy has come a long ways, it has not advanced fast enough and at the same time the rapidly rising complexity of systems and the introduction of multi-processor systems mean that this will never be the answer to the more general problem.

SystemC prototypes

The industry has spent considerable time and effort on the construction of virtual platforms based on SystemC. Examples of this are platforms created and proliferated by CoWare[4] and the proposed work project under Eclipse[5] (Virtual Prototyping Platform (VPP). These prototypes provide the hardware engineer with a flexible and adaptable platform on which bus traffic, power, performance and many other attributes of an implementation can be analyzed. These models are specifically intended for hardware architecture exploration. While much faster than the RTL prototypes already discussed, they are still at performance levels that keep them in the domains of hardware verification and firmware development. They are much too slow for software application development.

In addition, SystemC has failed to solve the model interoperability problem, something that the Transaction level Modeling (TLM) group of OSCI has been trying to rectify for quite some time now. Their latest attempt, which is still a long way from solidification, has failed to impress many in the industry, calling it “too little too late”. In addition, this proposed standard only addresses memory mapped interface which limits its ability to define a complete system level prototype.

Other companies, such as VaST Systems[6] and Virtutech[7] have forsaken the standards arena and used custom languages and tools to create faster models of the processors, memory systems and some aspects of the hardware. While they have successfully created prototypes with much higher performance, they suffer from the problems of model availability. Unless they write the models, it is unlikely that others will, making this more of a consulting and services model than a realistic tools business. The performance of such a complete system model is still not at a level suitable for applications development, but they have attempted to maintain as much cycle accuracy as possible, making it ideal for driver and OS development work, where such accuracy may be necessary. Another compromise is that these models assume working hardware and do not provide much in the way of visibility into the hardware models, making it less suitable for HW/SW debugging or system level analysis.

ESL Unifiers

As previously stated, one of the principle goals of proposed ESL flows has been to unify the design and development of hardware and software[8]. A unified flow would allow better design space exploration and enable tradeoffs in the domains of performance, power and flexibility. Most of the research into this area has concentrated on the automation of the hardware flow – following the lead of the migration to RTL where synthesis clearly became the central and most profitable aspect of the flow. But it is not clear that this will be the case for ESL. The application space is so broad that it appears to defy any single implementation path.

There is much more commonality in the needs for verification, even though there are multiple sets of users, each of which has a somewhat different set of requirements. The principle users are hardware developers and architects, the hardware verification team, system architects and the software team. The hardware teams and timing specific driver development tasks require models that closely match the implemented timing. Therefore the models must match or model the underlying architecture of the implementation. These users are adequately served by the solutions in place today and have already been discussed. The system architects – who are trying to define the optimal architecture for the hardware – do not need such detailed timing information, but they do need enough to be able to estimate performance and identify bottlenecks. This is a small market and one that few EDA companies have had success in pursuing, partly because nobody has really worked out how much detail and accuracy is needed. For the most part the software team, doing applications development, does not need much in the way of timing information. As each processor is clocked, time advances and so for each program thread, events will advance in the correct order. For multi-processor applications to work reliably, they must perform synchronization that does not depend on exact timing. Exact timing cannot be guaranteed in an embedded system, even when fully implemented, if any type of caching or resource sharing is used. If this does not include all systems today, then the addition of external events or timers can put systems into this class. Because of this, software should never be written to depend on specific timing and so a system level model for the software community can dispense with timing altogether, relying instead of sequential

order of execution and proper synchronization between threads. Synchronization is performed using semaphores, handshakes or a number of other mechanisms that ensure that software threads that need to communicate are in the necessary state for the exchange of data.

Changing Needs

Complexity

As time progresses, we are concerned not so much with the function of a block or an isolated algorithm, but of the control and coordination of many of these working together to form a complete, multi-function system. All this additional capability inevitably leads to complexity and it has been stated that complexity is primarily driven by communications. In other words, more functionality that is independent does not necessarily mean extra complexity, but then that does not create the same level of usability. It is when tasks start to coordinate and communicating with each other, that additional complexity is seen. In this regard it follows Metcalfe's law[9] in a strange way. Metcalfe's law of the Internet states that the usefulness of a network is proportional to the square of the number of users. In a similar way, total system complexity appears to be proportional to the square of the number of users, or in this case, independent nodes on that network. Each of these nodes is able to communicate with each of the others and to collaborate to perform the total function. Also by implication, each of those nodes are performing an independent task or coordinating with others to fulfill a more complex task. With the advent of multi-processor SoCs, software has now become truly multi-nodal since threads can execute in a fully concurrent manner and interact with each other in real time.

It has been shown many times that the human mind can only deal with a certain number of items at one time. While this number may vary slightly between individuals, it stays constant for each, independent of the abstraction of those items. Thus as the number of nodes in a system increases it quickly gets beyond the ability of any individual to keep track of what each is doing and the ways in which they communicate. Thus models are needed to encapsulate these behaviors and the communications between them and present this information to the user in a way in which they can easily focus on a few objects in order to analyze their behavior, performance or their use of resources. The other blocks may remain hidden unless they begin to impact the current investigation.

Software needs

It has often been said that software engineers do not need models or prototypes of the hardware. While this may have been true for the simple applications of the past, where cross compiling the code onto the host was quick and easy, it is most certainly not true for code that has been developed to run on multi-processors. Even though current desktops now have 2 or 4 processors, they provide a less reliable view into how the software would operate or perform on the actual embedded hardware, which may have

special communications between the processors, or require heterogeneous processors. They thus need a much more accurate prototype on which to investigate the communications and synchronization of this new type of software application. Consider a network-on-a-chip (NOC). With this kind of chip, there are many identical processors and attached to each processor is a communications router. These routers are then connected in a regular structure, such that each processor can communicate directly with its neighbors, but must go through multiple routers in order to communicate with others processors further away from it. This structure of the communications needs to be modeled and analyzed in order to locate possible inefficiencies or areas of congestion.

At the other end of the scale, many companies utilize physical prototypes to conduct software verification. While these operate at close to real time speeds and have very accurate timing, it is basically too late in the development cycle such that problems found in the software cannot be reflected by necessary changes in the hardware. This leads to inappropriate fixes in the software or compromises in functionality or performance of the complete system. With the introduction of multi-processor systems, this solution is also being tested in other ways since it is difficult to see what each processor is doing in real-time and operations such as single stepping are almost impossible. What is needed is a solution that provides the same level of performance, but is available early in the design cycle. This necessitates an abstract model of the hardware that can enable realistic execution of the software. However – it must be able to do this at close to real time speeds in order to be acceptable.

Accuracy and Timing

One of the most difficult things for hardware designers to get over is the level of timing accuracy necessary in a model. If for a moment we conceive of a situation where the hardware and software are going to be designed and developed alongside each other, then we must have models available long before the RTL has been written. The software engineers may need to look at the traffic patterns created by their algorithms and tweak the hardware or software to accommodate the needs. It may be decided that certain decisions that were initially made do not hold up to closer examination and need to be changed. This cannot happen if a lot of time and energy has already been in invested in the details of implementation. So if hardware has not yet been implemented, how can we know the exact timing? The only thing that can be shown is that it meets the requirements. That is one of the purposes of a system level virtual prototype – to work out the timing necessary and other details of the architecture long before they are frozen. Thus timing is an artifact of implementation, and the lack of timing in the early stages of a project does not imply inaccuracy, just the degrees of freedom currently left in the design process.

Instruction accurate

OVP[10] states that their processor models are instruction accurate. This statement requires a little investigation. In the realm of ISS models, they have usually been defined as instruction accurate or cycle accurate. The instruction accurate models are approximately

timed in that they claim to, on most occasions, execute each instruction using the correct number of clock cycles and they perform their I/O operations at sort of the right place within the instruction. The exact cycles on which things happen are not guaranteed. That is what is promised with cycle accurate models. However, very few cycle accurate models exist and most of them are really cycle approximate even though they claim otherwise. When OVP talks about instruction accurate, they mean this in the purely functional space and not in the behavioral space.

Function and behavior

This industry gets itself into so much trouble because of its loose use of terms and the above discussion is one of those. So please excuse a little digression. Functional models and behavioral models are strictly defined in the industry standard taxonomy[11]. To cut a long story short, a functional model does not include timing, although it may include sequence. A behavioral model includes timing although the level of detail of the timing is not defined. Both models may exist at any level of abstraction. Thus an ISS is a behavioral model. RTL and gate-level models are also behavioral models at different levels of abstraction. The OVP models are functional models. When they claim that they are instruction accurate, it means that the registers hold the correct values at the end of each instruction and create the right side effects from executing that instruction. They progress one instruction at a time and do not know anything about multi-execution pipelines, out of order execution or anything of those sorts. When an OVP platform is put together, each processor will advance one instruction at a time, as will any other processors in the platform. They may operate at different instruction frequencies, but the exact timing relationship between them has no conceptual meaning. When they attempt to use a shared resource, such as a memory, the model will arbitrate the access to the model and ensure that each transaction is conducted in a safe and repeatable manner. However, the order in which those access happen cannot be guaranteed. Thus if you have an application that depends on that level of detail, then an OVP model, or more broadly a functional model or even an approximately timed behavioral model is not for you.

Open Virtual Platforms (OVP)

The EDA industry has complained that the free availability of the SystemC simulator source code from the Open SystemC Initiative (OSCI) has fatally damaged the ability for commercial companies to make money from virtual platforms. This in turn means that few people outside of the Universities have much interest in extending or improving its capabilities. Any changes made to the OSCI kernel are treated with suspicion and customers are not willing to spend much for those improvements – be they speed or capabilities. In addition, SystemC failed to solve the model interoperability problem, something that the Transaction level Modeling (TLM) group has been trying to rectify for quite some time now as discussed earlier.

Imperas[12] has taken a different approach with OVP and OVPsim. First they have made the interface, which addresses the model interoperability problem, the primary entity that they are making public. Along with it, they are seeding the market with a number of models that demonstrate the capabilities of the interface. To build and debug models, a

simulator must also be available and Imperas has provided this, but only in executable form and only for the Windows platform. This free simulator is very much like the early version of Verilog that Cadence made openly available to everyone so they could develop models, interfaces or bolt-on tools for it. In the case of Verilog, the free simulator had dramatically lowered performance, but given that one of the prime motivations for OVP is performance, creating a low performance free simulator would be counter intuitive. The OVP website states that even higher performance is available from their simulator on other platforms, such as Linux, and that the free version does not contain features such as a multi-processor debugger – something that will be key to the development and verification of the next generation of platforms. It would thus appear as if Imperas has thought through the objectives and motivations for their donation of OVP.

OVP Basics

OVP is made up of four C interfaces: ICM for creating platforms, VMI for processors, BHM for behavioral blocks and PPM for peripherals. These interfaces are depicted in Figure 1.

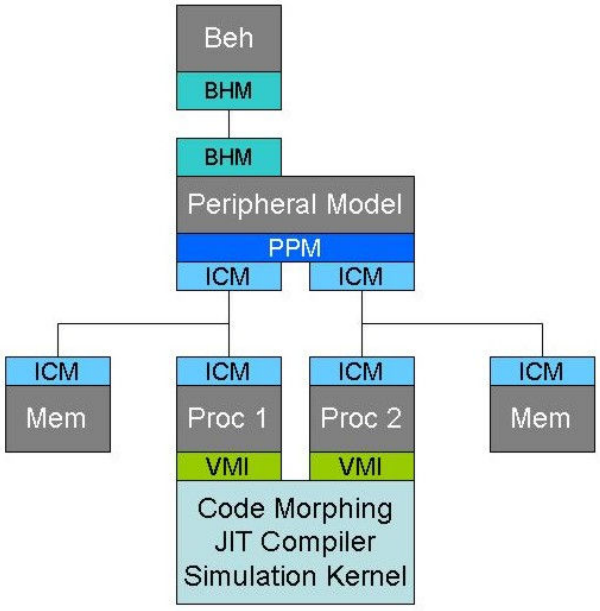

Figure 1: OVP Interfaces

ICM is the interface that ties together the basic blocks of the system, such as processors, memory sub-systems, peripherals and other hardware blocks. In this regard it is similar to the component construction that is done by SPIRIT[13] based tools. The difference is that SPIRIT provides meta-data about the blocks and the interfaces between them, but this itself is not executable in any way. The ICM interface is a C interface that when compiled and linked with those of each of the models and some object files produces an executable model. Given that it is based on standard C code, any C compiler can be used to create the model. Imperas has a commercial tool that will do the conversion from SPIRIT to the ICM model. In addition, the ICM interface allows

memory images to be defined for the start of simulation. That means that programs or data can be pre-loaded into the system model.

Processors in OVP are not ISS models. While the performance of ISS models has increased tremendously over the past few years, they are still limited to a million instructions per second (MIPS) at best and in many cases can be as slow as a few tens of thousands of instructions per second. This is too slow for software development and debug, which must be within about a factor of 10 of real-time. With embedded processors typically operating in the 1 GHz range, that means ISS models are orders of magnitude away from the desired performance levels. VMI is the processor interface, allowing the processor model to communicate with the simulation kernel and the other components of the system. In addition VMI is the heart and soul of the high performance execution provided by OVP. OVP uses a code morphing approach which is coupled with a just-in-time (JIT) compiler to map the processor instructions into those provided by the host machine. In between are a set of optimized opcodes into which the processor operations are mapped, and OVPsim provides fast and efficient interpretation or compilation into the native machine capabilities. This is very different from the traditional ISS approach which interprets every instruction. The software community has been using these techniques[14] for quite some time with impressive results and it appears that Imperas has done what we have seen many times in the past – that of migrating a successful software technique into the hardware design world. Another capability of VMI is to allow a form of virtualization for capabilities such as file I/O. This allows direct execution on the host using the standards libraries provided.

Extensible processors, such as those from Tensilica[15], are becoming a fixture in many SoCs. These call for a different type of processor model, where the core can be provided by the vendor, and the user has the ability to extend the instruction set to match the needs of their application. These extensions can also be added using the VMI interface. Thus it is possible to provide a base OVP model that is user extensible.

PPM, the peripheral modeling interface, is very similar to the fourth interface BHM, which is intended for more generalized behaviors. These models run in a second portion of the simulator that is called a Peripheral Simulation Engine (PSE). www.OVPworld.org states that “this is a protected runtime environment that cannot crash the simulator”. It does this by creating a separate address space for each model and restricts communications to the mechanism provided by the API. The principle difference between the two interfaces is that the PPM interface understands about busses and networks. It is thus similar in terms of functionality with the OSCI TLM interface proposal. The BHM is more like a traditional behavioral modeling language with processes activation, and the ability to wait for time or a specific event. This can handle more general forms of communications and thus provides the piece that TLM is missing.

OVP models the bus as a communications fabric but does not consider it to be a shared resource. In that respect all connections are pure one-to-one communications channels with their memories and peripherals. If bus arbitration was necessary, it probably means that you have multiple masters on a single bus, and you may need to model the bus as a

block in the system. However, this goes back to the previous discussion about why you would care about the exact traffic patterns at this stage in the design. Perhaps more important would be to see the general traffic levels on each bus, and be able to identify the peaks. These could then be investigated further to see what can be changed to even them out, or to decide on scheduling or priority issues as part of the implementation.

Playing nice with other simulators

Playing with legacy models is always a necessity when introducing a new environment. Normally, simulators tend to want to be masters and can call into other models or simulators. This creates a conflict when two simulators need to be bolted together because neither of them really wants to relinquish control. Imperas has taken the opposite stance saying that they are a slave and thus callable from other environments – such as SystemC. The reverse is not true. OVPsim cannot call a SystemC model, something that is perhaps quite natural since the calling of SystemC would bring the entire simulator performance back down to the very thing it is trying to replace. On the other hand, if someone has a SystemC platform today, substituting part of the system with an OVP model may bring about a large performance gain in relative terms. However, at the end of the day, Amdahl's Law tells us that we get diminishing returns dominated by the slowest running piece of the entire system, and thus even one SystemC model will make the system crawl along at the rates that we currently see and have been deemed unacceptable by the software development community.

Example

Several processor models, include those from ARM, MIPS and the OpenCores OR1K are available on the OVPWorld website along with prepackaged demos of them. One of the demos combines multiple heterogeneous cores together. Imperas has also made the free simulator available on-line so that other people can begin to create platforms or models of their own. Figure 2 shows the performance results that they obtained for each of the cores running a number of different benchmarks.

	OR1K			ARM			MIPS		
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS
linpack	5,783,952,671	13.11s	441	1,110,694,547	3.70s	299	97,571,640	0.51s	191
h264encode	20,655,155,698	48.60s	424	6,967,978,843	37.76s	184	21,324,418,192	56.99s	374
ucLinux			395*						
Queens				1,303,321,123	4.84	269			
Dhrystone	6,498,118,119	14.36s	453	1,214,070,280	3.75s	324	1,096,094,508	2.57s	427
Whetstone	140,023,166,816	228.36s	613	8,084,138,084	33.06s	244	2,589,794,515	8.65s	299
peakSpeed1	6,800,004,105	5.68s	1195	5,600,003,303	4.75s	1177	5,600,014,901	4.55s	1228
peakSpeed2	7,600,010,964	5.76s	1317	7,600,003,529	8.31s	912	6,400,015,068	5.37s	1187

Figure 2: OVP performance (on 3GHz PC host)

Conclusions

OVP has the potential to provide the first true system-level virtual prototype that is capable not just as a platform for hardware development, but also for software

development. That means it could be the first general purpose ESL modeling system that will form the cornerstone of complete ESL flows into both the hardware and software communities. While this has been done in specialized areas before, such as DSP design, it has never been solved in the more general case. Imperas has enabled, but not destroyed the commercial market for these prototypes, which means that it could get a lot more commercial attention than SystemC has garnered. If successful, it will also have addressed the model interoperability problem, and for that the entire industry should be thankful.

References

[1] Klein. *Hardware Software co-verification*. Mentor Graphics white paper. http://www.mentor.com/products/fv/techpubs/

[2] Harris, Stokes, Klein. *Executing an RTOS on simulated hardware using co-verification.* Mentor Graphics white paper. http://www.mentor.com/products/fv/techpubs/

[3] Andrews. *Managing design complexity thorugh high-level C-model verification.* http://www.mentor.com/products/fv/techpubs/

[4] Serughetti. Virtual Platforms for Software Development -- Adapting to the Changing Face of Software Development. Coware white paper.

[5] Eclipse Virtual Prototyping Platform (VPP) - http://www.eclipse.org/proposals/vpp/

[6] Hellestrand. Systems Architecture:
The Empirical Way - Abstract Architectures to 'Optimal' Systems. VaST white paper. http://www.vastsystems.com/docs/EmpiricalSystemsArchitecture20050722Pub.pdf

[7] Simics from Wikipedia. http://en.wikipedia.org/wiki/Simics

[8] Bailey, Martin, Piziali. ESL Design and Verification. A prescription for Electronic System Level methodology. Elsevier 2007

[9] Wikipedia, Metcalfe's Law - http://en.wikipedia.org/wiki/Metcalfe's_law

[10] OVP World website. http://ovpworld.org

[11] Bailey, Martin, Anderson. Taxonomies for the development and verification of digital systems. Springer 2005

[12] Imperas Inc website. http://imperas.com

[13] The SPIRIT Consortium. http://www.spiritconsortium.org/home

[14] John Aycock, A Brief History of Just-In-Time**.** ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 97–113

[15] Tensilica whitepaper: Configurable Processors: What, Why, How?
http://www.tensilica.com/products/WP_config.htm