Chip Design Magazine

By Neil Richards, James Green, William Stoye, and David Greaves

Published in **December / January 2004** issue of Chip Design Magazine

C Models Speed Co-Design

Virtual silicon meets the needs of hardware and software development simultaneously.

Developing system-on-chip (SoC) devices for DSL applications is a tricky business. These products usually consist of a highly flexible communications processor with multiple CPU cores, several external interfaces and multiple RAMs and are frequently integrated with DSL physical layer devices. GlobespanVirata (Redbank, NJ) is in the business of designing SoCs for DSL, and we know that the throughput, flexibility and timely delivery of the silicon and software depend heavily on close cooperation between the hardware and software disciplines.

Prior generations of these types of SoCs were developed using hardware emulation to provide early prototyping for verification. This process could produce a working model at about 1/100th of the speed of the final chip. An add-on board was then developed to provide interfacing to the two CPUs, and for external devices such as DRAMs. Subsequently, a special 1MHz PCI bus was constructed, which allowed the PCI interface, for one version, of the chip to be tested. Other interfaces (Utopia, for example) were attached to real devices, exposing them to real traffic. Ethernet MII and USB interfaces could be then tested by connecting to a specially constructed hardware board that acted as a proxy, relaying data between real (fast) hardware and the slower hardware of the simulation.

This overall approach could be very effective; a working model of the chip was available well in advance of tape-out date. The software team was able to boot the operating system on one CPU, pass network traffic over many of the interfaces, and verify the critical parts of the design, thereby getting an early start on driver development. As development times shortened, however, we found that slow simulation speeds hampered time to market. Additionally, crucial feedback often came late in the development cycle, off too late and the architecture could not be altered accordingly.

Establishing alternatives

With time, several alternative strategies have been developed, all of which are aimed at providing a view of the device thatâ€™s useful for both software and silicon designers earlier on in the design cycle. These include RTL simulation, FPGA prototyping, and C modeling

The RTL simulation option deploys HDL simulation tools on every software engineerâ€™s desk. While workable, the process can be prohibitively expensive (every workstation has to be equipped with a simulator) and integrating the procedure into the software development environment is extremely difficult.

Alternatively, designers can rely on FPGA prototyping to provide extremely fast simulation speeds, but designers also know that the process can require a lot of upfront work. Special care needs to be taken in partitioning and integrating into the development environment.

Lately the third approach, C modeling, has become of considerably more interest as it allows designers to build a â€œvirtual siliconâ€? model of the device in C. Itâ€™s true that some parts of the device have traditionally been modeled as part of the architecture development phase. But until recently, there has been no complete, working model of the overall chip available at this level. (see Figure 1)

Fortunately, things have changed for the better. These types of high-level models are now available and the advantages for the development teams are numerous: A large pool of non-RTL experts (members of the software team) can build and use the models, working in what for them is a familiar environment. The hardware development manager is happier; he can see a considerable increase in the capabilities of his verification team. Also, simulation times are likely to be shorter relative to RTL and gate simulations. And finally, integration with the software development tools is straightforward.

Overall, the approach is cost effective compared to emulation and is conveniently available earlier in the design cycle-particularly if delivered as part of the architecture development-and itâ€™s easy to share multiple models of the device among a very large software team

The process is not without disadvantages, however, as we discovered on an initial project wherein the system was tested. There wasnâ€™t enough time available to hand-build a full-chip C model from scratch. Also, there was considerable reuse of legacy silicon intellectual property (IP) on the project for which no C models were available. Therefore the IP required a considerable amount of effort to describe. And even after the IP models were developed, they frequently â€œdriftedâ€? as the IP vendors implemented changes without notifying us.

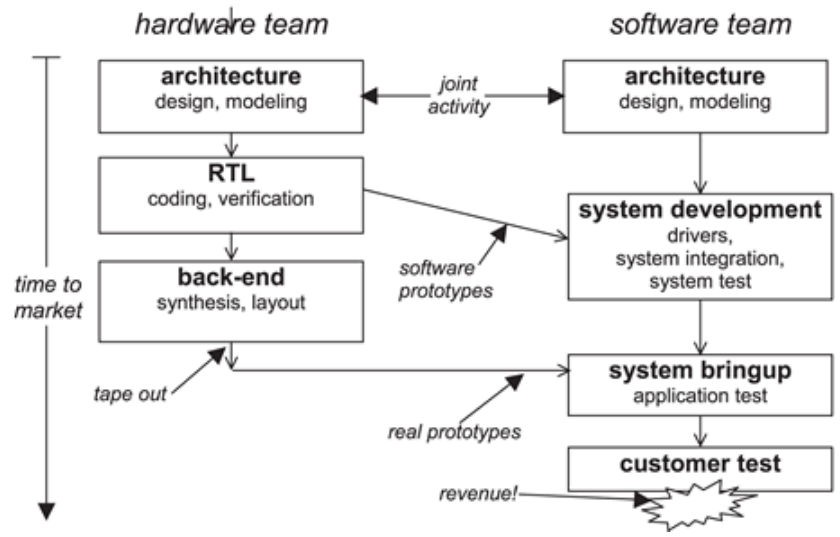

Figure 1: A system-modeling strategy should be used that permits continuous involvement of the software team in the overall SoC project.

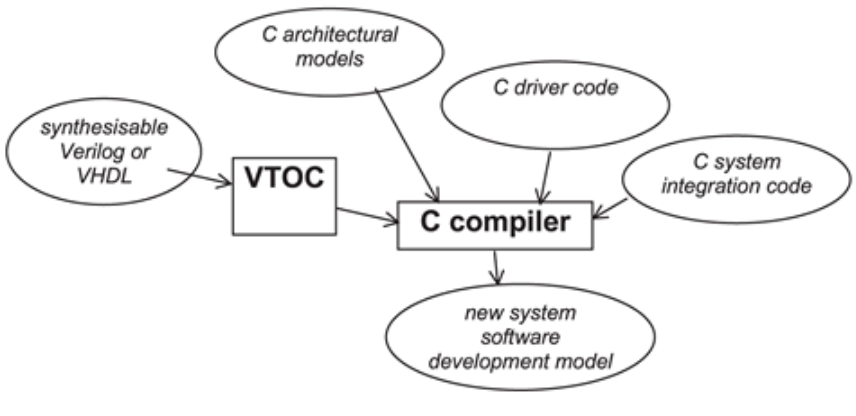
C architectural models
C driver code
synthesisable Verilog or VHDL
VTOC
C system integration code
C compiler
new system software development model

Figure 2: The VTOC Verilog/VHDL-to-C modeling allows the disparate hardware and software designs to be viewed as a single software entity.

Itâ€™s true that some C models of individual IP blocks did exist, which helped to finalize the design details for some blocks on this chip. However, these models were usually functional only (not cycle accurate). As a result, details of the interactions between blocks would be lost. Commercial tools which support C-based chip development had been explored, but the investment cost and time required to implement a full design flow precluded their use at this time

Applying commercial tools

Knowing the pitfalls as we did, we began to use tools VTOC from Tenison EDA (Cambridge, U.K.) to convert the synthesizable Verilog into C, C++, or SystemC, which provided a reliable, proven methodology for our project. VTOC performs a synthesis-like transformation of the input program, resolving the majority of scheduling decisions statically and resulting in a simple, straight-line representation of the execution of the Verilog program in each clock cycle. Since the output on our initial project had no scheduler, it was very easy to integrate with the C code that ran on the embedded device.

VTOC is entirely specialized to a 2-value simulation, with each Verilog vector corresponding to a similar C variable, so itâ€™s very easy for software engineers to interface with the tool and understand it. Our software team discovered that these transformations made the output code execute very quickly, as much as 10x faster than compiled Verilog-since the resulting code is doing less work than a full Verilog. (See Figure 2)

We found that VTOC can split the translation at Verilog module boundaries, so size itself is not an issue. However, some elements of the device required special attention: The outer layer of the chip including pads, clock tree, and metal layer sewing kit was stripped off, as none of these things contributed usefully to a cycle-accurate model. The main system clock generator for the chip was a phase-locked loop that generated a clock from a slower external crystal. This clock generator was not used and the system clock fed in directly from the external C++ testbench.

Each external peripheral was studied to see how it mapped onto the external world. The UART could be converted by VTOC, but the module within the UART containing the data shift register was replaced by hand-written C++ code. This process conveniently presented basic character input and output on the userâ€™s terminal. The USB and Ethernet physical layer circuitry had various analog elements and was removed. The Ethernet interface was attached at the MII interface to the appropriate debugging code, and in a later iteration was linked through to a physical Ethernet for some tests. The DRAM interface was attached to a simple hand-written model of suitable DRAM chips. This DRAM model was written in Verilog, and also passed through VTOC.

Within the digital logic of the chip, only the SRAM blocks and the CPUs needed any special attention. The SRAMs included a variety of single and dual ported memories, generated using a third-party tool. The Verilog provided by the generator was not fully synthesizable, and was replaced with trivial hand-written models. However, it was the CPUs that required the greatest attention. Once these special cases had been handled, the rest of the design compiled without any intervention. In fact, the vast majority of source files compiled without the need to even examine the RTL source.

The whole conversion process on the project was organized under Unix â€˜makeâ€™ so that no hand editing of any machine-generated files was required. We found that regenerating the entire model from the source RTL can take between 10 and 40 minutes, depending on the complexity of RTL changes.

Addressing disparate requirements

Helium 500 chip is the first project using this methodology. The design consists of two 32-bit CPU cores, on-chip SRAMs and caches, interfaces to external SDRAM, external peripherals, and multiple serial interfaces. Itâ€™s a mixture of hard macrocells, generated RAMs and synthesized logic. (see figure 3). The two CPUs are referred to as the Network Processor and Protocol Processor (the NP and the PP). The NP is programmed in assembler and performs low-level protocol operations. The PP is mainly programmed in C and C++ and performs higher-level protocol and management operations. In order to satisfy different engineering simulation requirements, two distinct strategies were used. These are referred to as the â€˜chip modelâ€™ and the â€˜hybrid model.â€™

In the chip model an instruction set simulator CPU model was used, written in C++ and inserted directly into the VTOC-generated simulation. This model provided a like-for-like replacement for the RTL of the processor core-which accurately represented the caches and the CPUs-so the load on the rest of the chip and the DRAMs could be shown to be cycle accuracy.

In the hybrid model, the high-level C driver code running on the PP was linked directly to the VTOC-generated C code, eliminating the need for the instruction set simulator. When the higher-level C code performed memory mapped-device read or write operations, these were trapped and used to trigger memory cycles in the VTOC chip simulation. Hence the higher-level protocol and management code executed at far greater speeds, than real-time. The NP was driven as a CPU simulation, even in the hybrid model, as it was programmed entirely in assembler.

The bulk of the VTOC simulation time was spent doing VTOC calculations. As a result, adding tracing was very cheap, even if applied to every cycle. We found tracing very useful, tracing many things and creating log files for everything. For every CPU instruction, we produced a register dump and traced every SDRAM access. This procedure gave us the ability to set up complex triggers when trying to analyze potential bugs. Similarly, software profiling and run-time checking were also supported in this environment.

On-chip RAM models were actually memory mapped files, so they could be examined and changed as necessary in an editor, even as the simulator was running-another important aid in the debugging process.

We were pleased to see that the VTOC environment also supported connection to external models, models not generated by VTOC. This was particularly useful in dealing with Flash memory devices, where a third-party provided model was connected to the VTOC-generated model of the external memory interface.

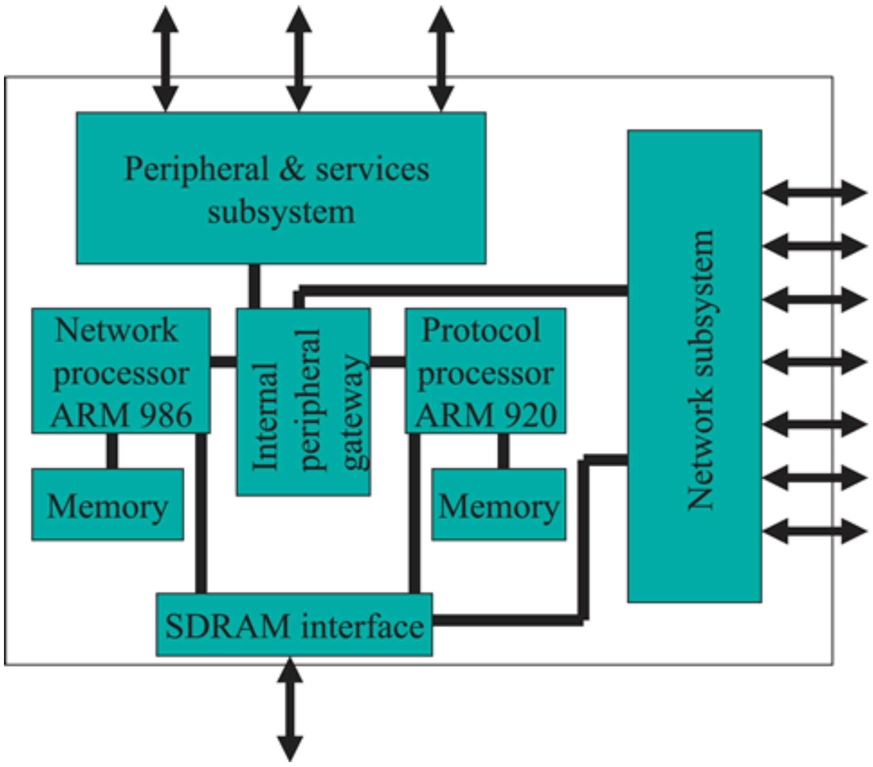
Peripheral & services subsystem
Network processor ARM 986
Internal peripheral gateway
Protocol processor ARM 920
Memory
Memory
Network subsystem
SDRAM interface

Figure 3: The design chip block diagram shows the major components in the design.

We found that it was also possible to connect the VTOC model to real hardware, external to the PC host. In the case of the Helium 500 development, this entailed both a serial ROM and an Ethernet interface.

We found the cycle-accurate model of the Helium 500 chip ran 4-5 times faster than the same RTL running on Verilog VCS. However, at the time, we were using VTOC models on an Athlon PC and VCS on a Sun UltraSPARC server. More recent comparisons with the current version of VTOC on identical machines suggest a typical 10-fold speed advantage over compiled Verilog simulation.

The hybrid model ran at the same speed, as measured in cycles per second. However, the results were more useful because the higher-level code executes without taking up any simulated cycles. Far more of the cycles were performing useful IO operations on the hardware model, than were spent executing simulated instructions for a CPU.

In order to speed the process, our principle technique was to simulate only portions of the chip. The makefile for the VTOC model allowed subsets of the chip to be created, with specific peripheral components excluded. This allowed simulation times of kilohertz simulation speeds to be achieved for specific software tests, depending on which peripheral elements were included.

As a cycle time, these speeds clearly were a great deal lower than the hardware the simulation replaced. Using a hybrid simulator, however, meant that all of the useful large-scale tests could still be performed. For example, the simulated operating system on the PP booted in approximately ten seconds, making it feasible to develop code as if on the real silicon. Meanwhile, the operating systems on the hardware emulation system required approximately 15 minutes to boot. An important benefit of all of this-executing simulations on inexpensive desktop PCs meant that every software engineer on the team could run jobs concurrently. This was a definite plus in reducing overall design time.

Saving the best for last

When a problem is being debugged, which must be traced through the VTOC-generated models, software engineers tend to use conventional C++ tools (for example, GDB). In addition to this, VTOC code can generate wave trace files in the VCD format. This is useful when characterizing specific issues, and discussing them between the hardware and software groups.

Software models are not as fast as hardware emulators or prototypes at pure cycles-per-second, but they are better in several ways for low-level software development. They are more deterministic than prototype hardware, and more transparent in their operation. For some software developers there is a barrier to overcome in terms of â€œtrustingâ€? the model, but this can be achieved with a suitable team structure.

All told, we were able to report a significant number of primary achievements on our initial use of VTOC. The RTL underwent extensive top-level and system-level verification using real applications software prior to tape-out. Numerous critical bugs were identified in VTOC and fixed in RTL, which would not otherwise have been found using conventional RTL simulation and verification. Additionally, the software drivers were ready to go when the chip was ready to tape-out

The best news for our software team came when first silicon arrived in the building. Within two hours, the operating system had been booted over the Ethernet interface, and routing and bridging operations successfully performed using software that had been developed during the chip implementation. This in turn, led to a faster delivery of working software to customers than had been achieved before by that same team on previous projects. Overall, we estimate that 3 elapsed months were saved over the lifetime of the project, in comparison with predictions based using previous methods.

Various system models were constructed using VTOC, including communication between multiple chips. One VTOC model was constructed with three identical chips attached to a PCI bus, with complete PCI cycles being tested between the simulated chips. We found that parallelism was easily exploited; multi-chip VTOC models were also run on multiple (cheap) PCâ€™s to improve simulation times.

The Ethernet interface in the software model was intercepted at the MII level, and programmed to read and write packets on a real physical Ethernet interface. Using this facility, the embedded softwareâ€™s bridge and router software was run in its entirety on the VTOC model, communicating with other IP hosts over the Ethernet port and exercising the embedded web server Using the chip-model simulation, the device was also booted over Ethernet from a real BOOTP server.

A VTOC model was used to measure the exact behavior of the DRAM interface under intense load from both CPUs, with typical application code rather than an artificial loader being used. Additionally, it was possible to perform â€˜what-ifâ€™ experiments on the DRAM controller to see the effects of changing certain interface parameters.

Future considerations

One interesting option to consider for future evaluation is to merge the use of VTOC-generated models with any C language models generated during the design phase of the project. The use of SystemC for system modeling during the design phase, for instance, may lead to higher-level models, which can in turn, be swapped in for one or more elements of the system. VTOC has an alternative mode that can generate SystemC rather than ordinary C++, so this line of study seems a promising area for future exploration.

Software and hardware programming languages have grown and evolved separately. We feel it to be unlikely that a single language will emerge to cover the entire spectrum of needs anytime in the foreseeable future. Given that fact, the use of language translation can help to allow hardware and software teams use the best tool for their specific tasks, and to work within environments that are familiar to each of them, while still producing results that work smoothly together. Using these translation tools, developers do not have to change the way they develop. They can continue to work in a natural manner, without any particular tool dictating a different, less comfortable approach. This is good news for both the hardware and the software world. î,©

Neil Richards is vice president Digital CSI design at GlobespanVirata, Inc.

James Green is a senior software engineer, also at GlobespanVirata.

David Greaves is the founder of Tenison EDA Ltd

William Stoye is CTO at Tenison EDA.