RTL-TO-C TRANSLATION CAN BE USED TO HELP VERIFY THE DESIGN OF COMPLEX SOCS BY LETTING GENERATED MODELS RUN PRODUCTION CODE AT HIGH SPEED. THE AUTHORS DESCRIBE THE TECHNIQUE USED TO REPLACE HARDWARE EMULATION DURING THE DEVELOPMENT OF A 5 MILLION GATE DEVICE FOR THE DSL MARKET

USING RTL-TO-C++ TRANSLATION FOR LARGE SOC CONCURRENT ENGINEERING: A CASE STUDY

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

In this article, we consider the problems faced by the hardware and software design team at GlobespanVirata working on the Helium 500 communications processor, a complex SoC design and typical of many chip designs today. A block diagram is shown in Figure 1.

GlobespanVirata develops complex SoC devices primarily for digital subscriber link (DSL) applications. These SoC devices generally consist of a highly flexible communications processor with multiple CPU cores on-chip, external interfaces, RAMs and often integrated with DSL physical layer devices.

This design consists of two 32bit CPU cores, on-chip SRAMs and caches, interfaces to external SDRAM, external peripherals and serial interfaces too numerous to mention. In terms of design styles, the project used a mixture of hard macrocells, generated RAMs and synthesised logic. Some parts were developed in-house for the project and others were reused from earlier in-house IP from previous devices. Some standard functions are third-party IP, some of which were known and trusted, and some of which were new and therefore viewed with the usual jaundiced suspicion of anyone with experience in this type of design. All in all, it was a typical modern-day application-specific SoC with the equivalent of 3 million to 4 million gates.

The earliest generations of the Helium family were developed using a hardware emulator to provide early prototyping for verification purposes. This system provided a working model at about one-hundredth of the speed of the final chip. An add-on board was developed to provide interfacing to the two CPUs for the product, and for external devices such as DRAMs. A special 1MHz PCI bus was constructed, from some very old components, which allowed the PCI interface for one version of the chip to be tested.

Other interfaces such as Utopia were attached to real devices, and real traffic passed over them. Ethernet Media-Independent Interface (MII) and USB interfaces were 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 emulation.

VERIFICATION TECHNIQUES

This approach was very effective. A working model of the chip was available well in advance of the silicon tape-out date. The software team was able to boot the operating system on one of the CPUs, and pass network traffic over many of the interfaces on the chip. In this way it was possible to verify critical parts of the design (good for the silicon team) and get an early start on the driver development (good for the software team).

The emulation-based approach has drawbacks. As back-end chip development times have shrunk, the "time-to-tape-out" advantage of the emulator has reduced, until it finally became a limiting factor that controls when the IP design can tape out. The emulation itself has an overhead. Significant hardware engineering effort was required to support the netlist and models for the emulator. Further, only one system could be supported, and so only one software engineer could use it at a time

Then there was the cost of the emulation environment itself. As newer generations of devices became ever more complex they required larger and larger emulation systems, with significant upgrade costs as a result.

ALTERNATIVES TO EMULATION

There are several alternatives, all of which are aimed at providing a view of the device for both software and silicon designers earlier in the design cycle. These are: RTL simulation; FPGA prototyping; and C modelling.

The first option considered was to deploy HDL simulation tools on every software engineer's desk and some experimentation was even done with this approach. There is one advantages to this: the environment seen by the software engineers is the same as that of the silicon designers. But that is about it. The disadvantages are numerous.

THE FIRST OPTION WAS TO DEPLOY HDL SIMULATION TOOLS ON EVERY SOFTWARE ENGINEER'S DESK

The engineering cost of doing this is high as RTL simulators are expensive and integrating the resulting tools with the software development environment is complex and difficult. Further, RTL design environments and simulators are unfamiliar to most software engineers.

Finally, there is the issue of poor simulation speeds. It is too slow for most software tests, and in any case would be no better than could be achieved by the silicon design team. The only way for software engineers to overcome this was to provide at least two workstations for each software engineer, and encourage many parallel tests. This in turn drives up the per-seat cost. On the other hand, the FPGA prototyping approach, that is, a do-it-yourself emulator, provides very fast simulation speeds. It is not as fast as the final chip but faster than with any other method. However, there were many practical problems.

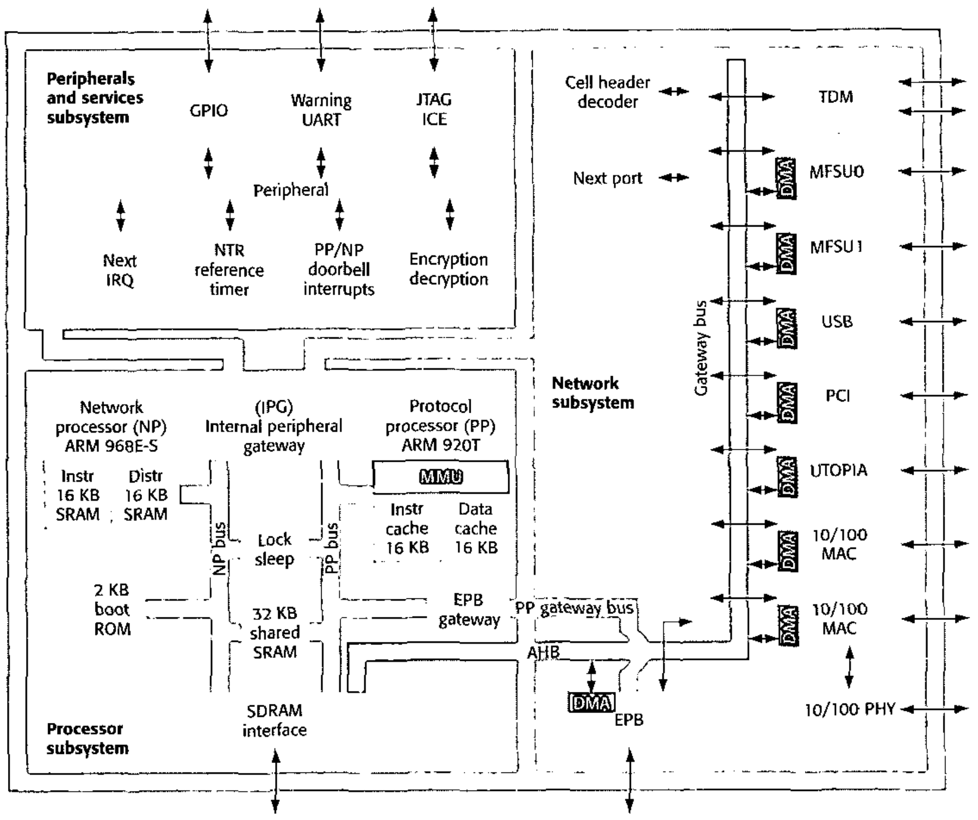
First, a PCB to hold several FPGAs would have to be developed and some of the CPUs and RAM blocks on the chip would have to be represented as external devices. In the FPGA environment, care needs to be taken with partitioning and integration with the development environment. On top of that, additional silicon design engineering effort is needed, particularly when combined with the effort required to support the resulting physical prototypes.

In the case of the Helium500 project, it was decided that within the relatively short timescale of the project it was not deemed suitable to rely only on developing a new FPGA flow for pre-tape-out verification. However, an FPGA based approach was used as part of the post-manufacture validation environment.

Of considerably more interest to us was the third approach: build a software model of the device in C. Some parts of the device had been modeled in this way as part of the architecture development phase, but there was not a complete working model of the whole

Farms of PCs can be used for multichip simulations

FIG. 1 BLOCK DIAGRAM OF THE HELIUM500



chip at this level. There are a number of advantages to the development team. First, on the team is a large pool of non-RTL experts, the software team, which is available to build and use the model, at least in theory. It is a familiar environment particularly for software engineers. The hardware development manager is happy with this because he sees a considerable expansion of the verification team's capability.

With the C model, the simulation times are likely to be very short relative to RTL and gate simulations and integration with the software development tools is straightforward. Further, the approach is very inexpensive compared to emulation and is available earlier in the design cycle, ideally delivered as part of the architecture development process. And, unlike emulation or an FPGA prototype, it is easy to share multiple models of the device among members of a very large software team.

The main disadvantage, at least on this particular project, seemed to be that there was not enough time available to hand-build a full-chip C model from scratch. Such models, if developed, may also "drift" as the IP supplier implement changes which are not tracked by the user.

ADDING TRACING IS VERY CHEAP EVEN IF THIS IS DONE EVERY CYCLE...AND WE TRACED MANY THINGS

The use of commercial tools that support C-based chip development had been explored but the investment cost and time required to implement a full design flow precluded their use during this project.

However, the advantages of C modelling appeared compelling. So, the question posed was: how can we quickly generate a C model of the device from the RTL model under development that supports cycle-accuracy, allows concurrent development of the software drivers and silicon, and extends the top level verification coverage of the chip?

RTL TRANSLATION

As a result of this set of challenges, the team at GlobespanVirata turned to ongoing research at the University of Cambridge, on the semantics of C and

Verilog. This work had produced a simple translation tool for converting Verilog into C with a number of desirable properties for solving this problem. As a consequence of this need the translator was much extended and became VTOC, now a product developed and sold by Tenison Technology EDA.

VTOC converts synthesisable Verilog into C++. It performs a synthesis-like transformation of the input program, resolving the majority of scheduling decisions statically and resulting in a straight-line representation of the execution of the Verilog program in each clock cycle. The GlobespanVirata engineers experimented with this, and found it suitable for building models for use by the software team. Since the output had no scheduler, it was easy to integrate with the C code that ran on the embedded device. VTOC is entirely specialised to a two-value simulation, with each Verilog vector corresponding to a similar C variable, so it was easy for software engineers to interface to it and understand it.

Since the resulting code is doing less work than a full Verilog implementation, it can go a great deal faster.

ISSUES WITH CONVERTING A LARGE DESIGN

Early experiments were successful, so the engineering team proceeded to translate an entire chip in this way, on a live development project, for the first time: the Helium500. This first happened in the summer of 2001.

The tool can split the translation at Verilog module boundaries, so size itself is not an issue. However, some elements of the device required special attention and these are described here. The outer layer of the chip, including pads, clock tree and metal-layer sewing kit, was stripped off. None of these things contributes usefully to a cycle accurate model. The main system clock generator for the chip is a phase-locked loop that generates a clock from a slower external crystal. This clock generator was not used, and the system clock was 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 presented basic character input and output conveniently on the user's terminal.

The USB and Ethernet physical-layer circuitry had some analogue elements and were removed. The Ethernet interface was attached at the MII interface to suitable 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.

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, created by a third-party generator product. The Verilog provided by this generator was not fully synthesisable, and was replaced with trivial hand-written models. The CPUs needed the most attention.

CPUS AND DRIVER INTEGRATION

The Helium500 contains two CPUs, referred to as the network processor and protocol processor (NP and 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. Although this arrangement is specific to this device, the solutions described here would apply to other cases. In order to satisfy different engineering simulation requirements two distinct strategies were used. These are referred to as the chip model and the hybrid model.

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

In the hybrid model, the high level C driver code running on the PP is linked directly to the VTOC-generated C code, removing the need for the instruction set simulator. When the higher-level C code performs memory-mapped device read or write operations, these are trapped and made to cause memory cycles in the C-code chip simulation. This means that the higher-level protocol and management code executes at far greater than real-time speed. The result is not fully cycle-accurate, but allows a great variety of more interesting tests to be run. The NP was driven as a CPU simulation, even in the hybrid model, because it was programmed entirely in assembler.

TRACING AND SPEED

The bulk of the C simulation time is spent doing VTOC calculations. As a result, adding tracing is very cheap even if this is done every cycle. We found tracing very useful, and in practice we traced many things. For ➔

example, for every CPU instruction we got a register dump and we traced every SDRAM access. This gives the ability to set up complex triggers when trying to analyse potential bugs. Software profiling and run-time checking are also supported in the environment. On-chip RAM models were memory mapped files, so they could be examined and changed while the simulator was running, another aid in debugging.

The VTOC environment supports connection to external models, that is, models not generated by the tool. One such example is the flash memory devices used, where a third-party provided model was connected to the VTOC-generated model of the external memory interface.

It was also possible to connect the VTOC model to real hardware external to the PC host. In the case of the Helium500 development this was achieved for both a serial ROM and an Ethernet interface.

SOFTWARE MODELS ARE NOT AS FAST AS HARDWARE EMULATORS BUT THEY ARE BETTER FOR LOW-LEVEL SOFTWARE

The cycle-accurate chip model for the Helium 500 chip ran on a 1.6GHz Athlon PC at about 150Hz. In comparison, the same RTL running on Verilog VCS on a 700MHz UltraSparc server ran at about 38Hz. The generated C code runs faster because it does less: it is a two-value simulation, and all scheduling has been determined at compile time.

The hybrid model runs at the same speed, as measured in cycles per second. However, it gives more useful results because the higher-level code executes without taking up any simulated cycles. So, far more of the cycles are performing useful I/O operations on the hardware model, rather than spending its time executing simulated opcodes for a CPU.

In order to go faster, the main technique used was to simulate only portions of the chip. The make-file for the VTOC model allowed subsets of the chip to be created, with specific peripheral components excluded. This allowed simulation times of 600Hz to 1000Hz to be achieved for specific software tests, depending on which peripheral elements were included.

As a cycle time, this is a great deal less than the hardware it replaces. Use of the 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 around ten seconds, making it feasible to develop code as if on the real silicon. On the hardware emulation system mentioned earlier, the operating system took around 15 minutes to boot. In addition, execution on cheap PCs meant that every software engineer could run it concurrently.

The VTOC compiler itself was not used by every software engineer. Instead, a core 'conversion team' was responsible for providing working models to the software group. The conversion team accessed the latest RTL from the source control system, ran VTOC, and did any necessary interfacing with the hardware team. This structure worked because the learning time for all was minimised, and the load on the hardware team kept to a minimum.

The conversion team consisted of two software engineers. The initial effort required on this project was about three man-months before concrete results were obtained. Much of that time was spent learning the VTOC tool and about the hardware design being converted. On subsequent projects, this time should be greatly reduced.

The conversion team remained in existence after this initial time as a focal point for interactions between the hardware and software groups, and to provide incremental improvements, such as the building of chip models with selective peripherals removed, as required by the software team.

When a problem is being debugged which must be traced through the VTOC generated models, the software engineers tend to use conventional C++ tools, such as the GNU debugger. In addition to this, the generated C code can generate wave trace files in the VCD format. This is useful when characterising 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 'trusting the model', but this can be achieved with a suitable team structure.

RESULTS

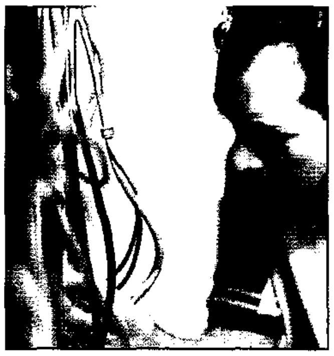
The primary achievements of the use of generated C models on this project were that the RTL had received extensive top level and system-level verification using real applications software prior to tape-out. As a result, some critical bugs were identified in the C models and fixed in RTL which would otherwise not have been found using conventional RTL simulation and verification. At the end of the project, there were 35 hardware bugs in the bug database that were originated due to the use of the models. And the software drivers were ready to go when the chip was ready to tape-out

When first silicon arrived in the building, within two hours the OS had been booted through the Ethernet interface, and routing and bridging operations performed using the software that had been developed during the chip implementation. This in turn led to faster delivery of working software to customers than had been achieved by the same team in previous projects. Overall, we estimate that three elapsed months were saved in the lifetime of the project, compared to the likely timescale if previous methods had been used.

It is worth noting that the ability to trace and display timing waveforms from the C simulations greatly aided communication between the software and hardware teams, allowing confirmation of suspect behaviour to made much more quickly. The C simulations also highlighted some inefficiencies in the hardware implementation that were subsequently improved by the silicon design team before final tape-out.

Various system models were constructed using the VTOC environment, including models of communication between multiple chips. One was constructed with three identical chips attached to a PCI bus, with complete PCI cycles being tested between the simulated chips. Parallelism is easily exploited: multiple chip-level models were run on multiple, cheap PCs to improve simulation times.

Separate models were built with four chips attached to a Utopia bus, and two chips attached to a TDM bus. The Utopia simulation allowed software to be optimised for maximum network throughput, and performance measurements to be calculated before real silicon was available, which was useful for sales and marketing at the very least.



PARALLELISM IS EASILY EXPLOITED: MULTIPLE CHIP-LEVEL MODELS WERE RUN ON MULTIPLE, CHEAP PCS 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 physical Ethernet interface. Using this facility the embedded software's bridge and router software was run in its entirety on the C model, communicating with other Internet Protocol hosts through the Ethernet port and exercising the embedded web server. Using the 'chip model', the device was also booted through Ethernet from a real Bootp server.

A generated C model was used to measure the exact behaviour of the DRAM interface under intense load from both CPUs, with typical application code rather than an artificial loader being used. It was possible to perform 'what-if' experiments on the DRAM controller to see the effect of changing certain interface parameters. One final advantage of this approach, although not exploited on this particular project, is that the models can be made available to key customers prior to silicon availability, allowing customers to accelerate the development of their application software as well.

FUTURE WORK

There are pointers to future savings in effort. The most obvious case is that bought-in models of CPUs and other standard components would speed the whole process and reduce the total effort required in any one development. The integration of the hybrid simulation with the CPU model and the operating system is also an element that could be carried over between projects.

What additional benefits are possible with this method? The most likely one appears to be the continued use of the resulting models after the device is available. This can help ongoing software development through being deterministic and more testable; through greater transparency of access, such as access to internal signals; and through increasing ease of software development by other sites and companies, by removing the need for low-volume developer versions of the hardware.

Another interesting option for future work 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 modelling during the design phase, for instance, may lead to higher-level models that can be swapped in for some elements of the system. VTOC has an alternative mode that can generate SystemC rather than plain C++, so this seems a promising area for future exploration.

The results were earlier availability of driver software, increased application-level verification before tapeout, and better understanding between the hardware and software groups.

The use of language translation can help to allow hardware and software teams to each use the best tool for the job, and work within environment that are familiar to each of them, and yet produce results that work smoothly together. The developer does not have to change the way they develop, and avoid tools dictating a different approach. ■

Dr William Stoye is CTO of Tenison Technology EDA. David Greaves is a lecturer at the University of Cambridge Computer Laboratory and the founder of Tenison Technology EDA. Neil Richards is director of IC design at GlobespanVirata. James Green is a senior software engineer at GlobespanVirata.