C for System Level Design

Guido Arnout
CoWare, Inc.

Abstract

Few people disagree with the fact that today about 80% of a system is software running on a "platform" of general purpose or custom processors (CPU and/or DSP) tightly coupled with unique dedicated hardware. This makes C (or C++) an obvious candidate for a system level design language. Without good hardware/software partitioning tools and support for C-based hardware design, the software content may have to increase by necessity. With the right hardware support a system team has the flexibility to make cost, performance, power trade-offs and decide later in the game how much of the system is software and how much is hardware. Another issue is legacy software and hardware. Legacy C software is well understood but legacy hardware is usually only available as RTL (Verilog or VHDL) at best. Therefore the ideal system level design language is C (or C++) based, accommodates hardware design but also co-exists with the vast legacy of Verilog and VHDL based re-usable hardware. CoWare N2C is practical solution, used in real life design around the world, that a) preserves the C software development paradigm for software people, b) adds the necessary clocking to C to enable hardware designers to move C functionality into a hardware architecture, and c) co-exists (for co-design and co-simulation) with existing hardware in Verilog or VHDL.

Defining a system from a napkin

Q. What do system designers do first when they design a new system or update an existing system ?

A. They start with a napkin sketch: an interconnected set of functional blocks that represent the system they want to build. Between this napkin and the first analysis of the system are numerous chapters of ambiguous text, often in English to facilitate system design with team members across the globe, and numerous prototype boards.

Q. What do system designers want to do when they define a system ?

A. They want to execute the system specification to make sure that the system they specified behaves the way they expect it to behave. Currently they have to wait until part of the system, usually the hardware, is fully designed and available as a prototype board.

CoWare N2C models a system as a number of interconnected functional blocks that can be simulated in which each functional block is an autonomous process. The system specification is driven by a testbench, which is a model of the environment the system lives in.

There is currently no lack of attempts to come up with a new system level language that extends to the system from the hardware up.

In contrast, CoWare took a top down approach that reuses existing languages as much as possible and only adds the necessary ingredients. Instead of defining a new language or extending the C or C++ language, CoWare has defined a **language independent** layer to specify interconnected autonomous functional processes. Each functional block can be specified in its appropriate language that can change as it evolves in the design process from system specification to fully implemented system.

Re-using existing languages

In CoWare N2C, a functional block can contain:

- a C level system model, new or re-used, that after hardware/software partitioning either represents software or hardware
- a C level system model, new or re-used, that represents a software function, a logic function or an analog function
- a C level model, new or re-used, that represents part of the environment the system to be designed must live in
- a Verilog level model, new or re-used that by the choice of language represents hardware

- a VHDL level model, new or re-used that by the choice of language represents hardware

Adding multi-threading

Q. Since C is the dominant software language, Verilog and VHDL evenly split the hardware world, and any analog function can be written in C, can any system be described re-using existing languages ?

A The language independent layer around interconnected blocks defined in existing languages is insufficient since this does not include the notion of concurrency. Only Verilog and VHDL model concurrent processes but in embedded systems, it is important to be able to tell the real time operating system how different software functions can be scheduled or in case of multiple processors, which software functions should be executed concurrently.

CoWare therefore added the ability to define **independent threads** in a language independent way. This is helpful for the software side and essential to model hardware functions in C to be translated to RTL later.

Using C for hardware design

Q. How can I use C to describe hardware if I can only define pure functional behavior ?

A What is missing is the ability to define clocks and to add cycle information to C.

To enable a designer to define a hardware architecture in C, CoWare added the ability to define multiple asynchronous clocks in C and to add clock cycle information as a wrapper around C functions or embedded in the C code.

With the exception of timing at a finer granularity than clock cycles, this makes it possible to write anything that can be written in Verilog or VHDL at the C level in CoWare N2C.

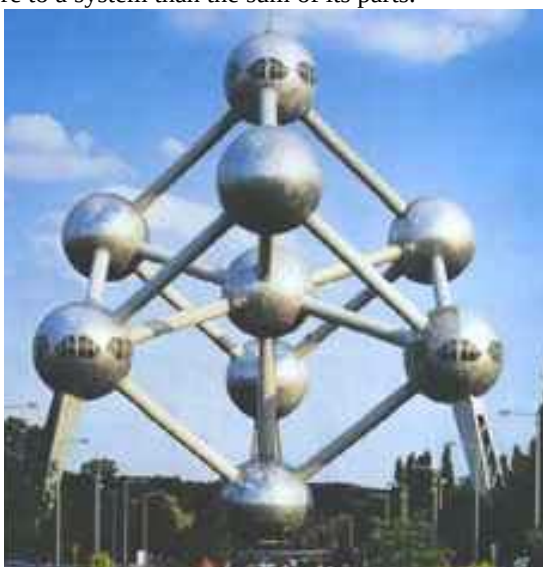
The main advantage of this is that it is now possible to refine any C level function, that is targeted for hardware implementation after hardware/software partitioning, through the addition of clocking and clock cycles, into a lower hardware-oriented C level that can be translated directly into Verilog or VHDL.

The ability to write the equivalent of Verilog or VHDL in C:

- enables designers to stay at C level for most of the implementation cycle
- makes it possible to move certain functions that now have to be implemented in hardware over time to software as faster and faster processors become available
- dramatically accelerates simulation time since C code executes much faster than Verilog or VHDL code.

CoWare Interface Synthesis™

The Atomium, Belgium's contribution to the World Fair of 1958, is a useful illustration to show that there is more to a system than the sum of its parts.



Each sphere may have different content but there is also content in the tubes that interconnect the spheres. More importantly, the tubes give the system its overall structure and its meaning beyond being merely nine spheres. A system is, after all, an interconnected set of functional blocks that may or may not be written in different languages. This is a good way to look at the CoWare N2C approach.

In CoWare N2C, the behavior of blocks is, as in the Atomium, clearly separated from communication between the blocks. This makes a lot of sense. Imagine that the escalators in the tubes that connect the exhibition halls in the spheres are embedded in the spheres. The Atomimum would be a pile of balls and not be "reusable" as an exhibition hall and conference center today.

This separation of behavior and communication also enables the synthesis of the implementation of a communication protocol into its low level software drivers and the interface logic between software and hardware components. It also enables hardware to hardware interface synthesis.

This represents tremendous advantages:

- it allows a system designer to think system all the time without being dragged down to the details of the processor(s) used in the system.
- it enables a much higher degree of IP re-use since the communication details no longer need to be blended in with the rest of the functionality of the IP. In essence, it makes the IP independent of the processor or bus used in the system
- it eliminates a lot of the tedious and error prone system level integration that, some analysts claim, consumes as much as 40% of the design time
- it empowers design teams to quickly repartition a system to explore different architectures or to take advantage of new IP in design derivatives

Conclusion

We live in a world full of talk of IP re-use. This philosophy should not only apply to functional IP or processors but also to languages involved in the description of functional behavior.

CoWare N2C has proven that a language independent definition of interconnected functional blocks, which re-uses existing popular languages for hardware and software, enables efficient capture of today's complex embedded systems. When combined with interface synthesis and the necessary extensions to the C language to enable hardware design in C, it can lead to a dramatic reduction in time-to-market.

CoWare N2C enables true IP re-use. It implements the strategy of the System Level Design Workgroup of VSIA and offers a practical solution to what the System Level Design Language committee (SLDL) attempts.