High-level synthesis, verification and language

John Sanguinetti, CTO of Forte Design Systems Inc. - February 22, 2010

Abstract
The preferred high-level design methodology proceeds from high-level code to RTL code. Good verification practice requires that the input to High-level Synthesis (HLS) be verified first, via simulation (or some other analytical means), and then the output of HLS be verified, again via simulation or some other means. Using SystemC as the input language to HLS enables this flow, but using C as the HLS input language imposes a serious limitation on doing verification this way.

Introduction
High-level design has many advantages over the more commonplace design flow which begins with RTL code. Among the most compelling advantages is the improved verification efficiency which a higher level of abstraction offers. It is apparent to the point of being self-evident that when the source code of a design is created, there will be fewer errors if the source is at a higher abstraction level than if it is at a lower level. However, there is still a process required to verify the transformations which are applied to the design description as it proceeds through the design flow from creation to final realization.

Figure 1 shows the initial steps in the High-level Design (HLD) flow, without any of the verification steps shown. There is more than one way to place the verification steps in this flow, and the choice of where they go is heavily influenced by the high-level design language that is the input to the RTL Creation, or High-level Synthesis (HLS) step.
Figure 2a shows how the verification steps fit if the HLD input language is "Plain old C" (PoC) - ANSI-C or C/C++ - and figure 2b shows where the verification steps fit if the HLD input language is SystemC. There is an additional verification step in the SystemC flow, which is labelled a validation step. For our purposes, verification means demonstrating that the design functions correctly. Validation means demonstrating that the design has the same functionality as a previous version of the design.
C vs. SystemC for design input
As ESL tools and methodology have developed, there are essentially two approaches which have achieved some level of acceptance — see Figure 3a and 3b. Most algorithms have a C representation, which can either be fed directly to a HLS tool (Figure 3a) or converted to SystemC, and then fed to HLS (Figure 3b).

While intuitively it is appealing to use C input directly for HLS, in practice that turns out not to be the best method. It is seldom the case that C source code can be synthesized into the desired hardware architecture directly. A substantial amount of human input to the hardware design process is required, even when using HLS tools. As shown in Figure 2, there is an iterative design optimization step between architecture design and RTL creation. Whenever a change is made to the design source code, it must be verified. As can be seen in Figure 2b, the verification step after Architecture Design is direct. When the HLS input source language is PoC (Figure 2a), the verification step after Architecture Design takes place after RTL creation (HLS), and is done on a transformed version of the design. This is much more ungainly.

Current verification methodology between RTL and gate levels is the right model
When using the standard RTL design methodology, verification is done directly on the RTL source code. This is the obvious way to do things. The RTL code is simulated or otherwise analyzed, changes are made, and verification is repeated. When that is done, logic synthesis is applied to transform the RTL to gates, and the resulting system model is verified using either simulation or formal equivalence methods. It is important to note that a gate level simulation uses the same test bench as the RTL simulation. In a system with multiple modules which interact, the process is shown in Figure 4.
The RTL source code describes the behavior of the communication interfaces between modules directly. As a result, simulating multiple modules together using the RTL source gives a faithful representation of the system's behavior. Verifying the behavior of the system design can be completely accomplished at the RTL source level. The only verification required at the lower gate level is that necessary to demonstrate the validity of the RTL-to-gate transformation. **The same methodology is the preferred one for the next higher level**

When using a high-level design methodology, the verification problem is the same, but verification is done at the higher level. Again, it is obvious that the most effective way to verify a design is to do the analysis at the level of the design's source, which in this case is the input to the high-level synthesis program. The high-level source code typically is verified by simulating the system model as shown in **figure 5**.

The important thing to notice in **figure 5** is that the system model can be verified because the interfaces are accurately modeled with SystemC. In SystemC, interfaces can either be timed, meaning they are cycle accurate, or they can be untimed. Untimed, or TLM, interfaces are abstractions of timed, or cycle-accurate, interfaces. In general, to adequately verify a system model, you need to simulate the model with timed interfaces, since system behavior can be heavily influenced by the functioning of the interfaces, and interfaces can be sensitive to timing. The important point is that all of this verification can be done before producing RTL with HLS.

There is an additional step when using a SystemC methodology, which is validation of the RTL produced by HLS. Once the SystemC system model has been verified, HLS is run to produce RTL, and that RTL must be shown to be a correct transformation, in an analogous way to the validation step between RTL and gates after logic synthesis. There are two ways to validate the RTL, simulation and sequential equivalence checking. Simulation, the more general method, is simply done by plugging the RTL modules into the system model and running the simulation model using the same testbench as used at the higher level. Sequential equivalence checking is a formal technique which, under certain circumstances, can determine that the RTL is functionally equivalent to the HLS input. It is important to note that the validation step is a much more limited activity than functional verification performed on the HLS input.

**SystemC allows interfaces to be timed, PoC does not**

This highlights a crucial distinction between using PoC as the high-level design language and using SystemC. Since PoC has no capability to capture the function of an interface, the system model cannot be simulated at the system source code level. The verification process when using PoC is shown in **figure 6**.
Here, the C synthesis tool is applied before the system model can be simulated, and the system model is simulated with RTL modules. Simulation at RTL is both much slower than ESL simulation and it is much more difficult to relate an error to the original cause of the error in the source code. This is a very ungainly verification process.

The example shown in Figure 7 is a producer-consumer, with a shared memory between the two. The shared memory consists of two memory banks, each containing 32 elements. The two processes are threads in the design. There is an arbitration mechanism which allows exclusive access to one thread at a time. The producer writes 32 data elements into bank 0, then does the same with bank 1. The consumer reads 32 elements bank 0, then repeats with bank 1. Each process only does 4 memory accesses at a time before releasing its buffer and then reacquiring it. This implements a simple producer-consumer using a ping-pong double buffer.

The arbitration is done by an acquire() method, the implementation of which is not shown. There is an acquire method for each port, p1 and p2.

This simple interaction between two processes and a memory can be simulated using SystemC and the code easily debugged. However, consider the case if this example is written in PoC. There is no mechanism in C to support concurrent processes, and no notion of time. Consequently, the acquire method which causes a process to wait until a condition is satisfied cannot be implemented directly. The end result is that either the arbitration process is modeled in a unique, model-dependent way, or it is modeled in the C source but the execution takes only one path. Often it is simply not modeled in the C source, but added in during synthesis.
The producer and consumer threads are shown in figure 8, but the code has a bug in it. The two loops in each process should be the same, but the limits of the outer and inner loops of the consumer are reversed from those in the producer. This is a simple mistake that causes incorrect behavior of the system. When simulated in SystemC, the erroneous behavior is evident and the bug can be fixed before the system is synthesized. When executed in C, the bug would not be evident, since the release/acquire sequence would not result in a change of control from producer to consumer. Hence, this bug would not be found until after synthesis produced RTL which could be simulated.

Conclusion
Verification should take place as close to the original source code as possible. In the context of ESL design, that means that simulation should be done before high-level synthesis, not after. However, language limitations force verification of inter-module communication to be done at one level or the other. With SystemC, verification can take place before high-level synthesis. With Ansi-C/C++, verification must be done after synthesis. This is a serious limitation.

We can summarize with the dictum "Synthesize what you simulate". Simulation is the verification step. It is vitally important to verify the design before performing high-level synthesis, not after. It is not effective to synthesize a design which does not work.

About the author:
John Sanguinetti is Chief Technology Officer of