Soft Copy

Test & Measurement, DAQ

Software

Interfacing VHDL and Verilog Designs to C++ Models

by Donna Mitchell, SynaptiCAD

C++ models add many new capabilities to Verilog and VHDL simulations including the ability to use high-level data structures, constraints and random data generation. C++ models are also very useful for co-simulation of hardware and software. Despite the advantages of C++ models, they have been relegated to simulation of large systems because of the cost associated with setting up the C++ environment. New techniques are now available to automate the process so that C++ models can by used by anybody. Using graphical code generation tools and public domain C++ libraries, engineers can set up a C++ environment and start simulating in just a few hours.

C++ Communication with HDL

Communication between C++ models and HDL simulators is usually achieved through a programming language interface that allows events and data to be transferred between the environments. For Verilog simulators, the Verilog PLI interface is used. For VHDL simulators, the method varies with each simulator; however, they work in essentially the same way as the PLI interface.

Control of a simulation begins in the HDL simulator. The C++ model registers itself with the HDL simulator and asks to be notified when certain events happen. The HDL simulator runs until a registered event, and then it stops and hands control over to the C++ model. The C++ model reads signal and data values from the HDL simulation, performs behavioral actions, writes results back to the HDL simulation and returns control to the HDL simulator.

The main challenge using PLI type interfaces is creating a method for automating the connection between signals in C++ and HDL models. Often several thousand lines of PLI-based code are required to manage the interface between the environments. This means that the user must master C++ object-oriented programming techniques and become proficient at building libraries in order to get things done efficiently. It is also difficult to develop a set of classes able to handle hardware concepts such as the passage of time, parallelism and signal driver resolution. Fortunately, C++ modeling libraries and GUI-based modeling tools can help users to overcome the most demanding of these challenges.

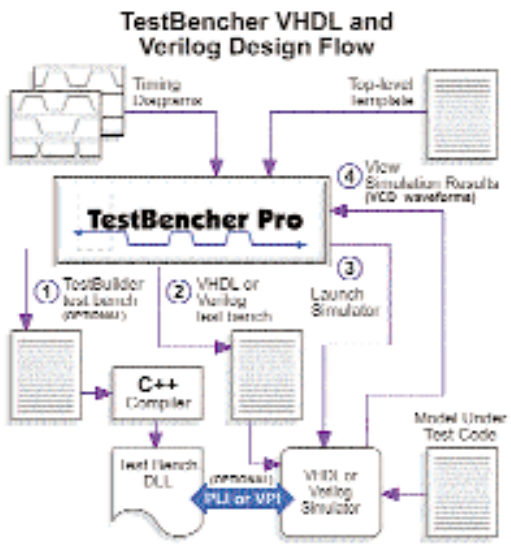

Figure 1. C++-Based hardware design flow.

C++ HDL Libraries

Several open source libraries such as SystemC and TestBuilder provide a framework for C++ hardware modeling. SystemC v2.0 provides the modeling constructs for both high-level behavioral models, gate level design and links for mixed C++ and HDL simulation. TestBuilder is a verification library that provides transaction-level modeling features, constrained randomization and automated signal mapping between C++ signals and HDL signals.

Working with C++ libraries provides an advantage in developing code; however, the initial learning curve is still significant. The syntax and library function calls are different for each library that is incorporated into the design. There is often a framework and recommended way to structure the design when using the library. In addition, users must learn how to set up the C++ compiler and HDL environment so that C++ library code is properly linked with the simulator. To make matters even more tedious, compilers on different platforms often require different options (for example, different make files). Graphical code generation and GUI-based project management tools address these issues by automatically generating model code, and they make files that are portable across platforms and different simulators and compilers.

Automatic Code Generation

GUI-based code generators can completely automate the mapping code between C++ signals and HDL signals and keep these mappings up-to-date as changes are made to the design source code. Similarly, these tools can hide from users C++-specific details such as syntax requirements for data structure definitions and template instantiations. SynaptiCAD's TestBencher Pro tool is one such graphical code generation and GUI-based project management tool. TestBencher generates bus-functional models for several languages including SystemC, TestBuilder, Verilog, VHDL, e and OpenVera. Users draw language-independent timing diagrams to describe each bus transaction graphically, and they enter information about the relevant data structures including random data generation and packing order. The tool generates a bus-functional model directly from this information.

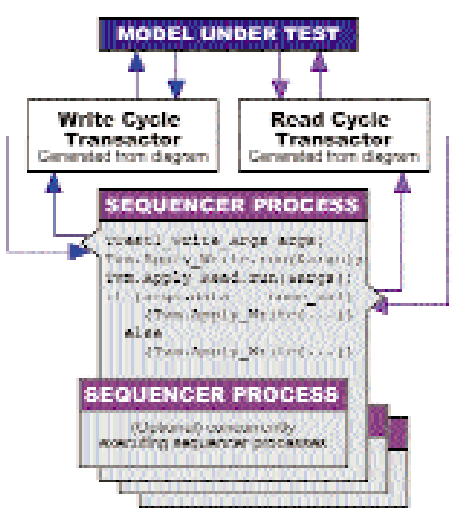

Figure 2. The TestBencher Pro's sequencer process controls and monitors the execution of the timing process.

GUI-based project management tools can be used to handle all of the external compiler and simulator control necessary to build C++ dynamic libraries, link them to the HDL simulator, and run the simulation. For example, TestBencher can launch simulators and compilers automatically, and hand off the generated code so it is very easy to move from test bench development to simulation. It can also import the final simulation results so that they can be viewed graphically. Figure 1 shows a typical modern design flow for C++-based hardware modeling.

C++ Golden Reference Models

In addition to using C++ to develop design models and test benches, C++ can also be used to develop reference models that run in parallel with the design model and provide another type of check for the design. Golden reference models are high-level descriptions of a design, and they are used to compare to the results of an RTL-level model during simulation. Reference models usually model interaction between components at the transaction level (for example, read transaction/write transaction) instead of at the signal level. When the reference model is created, the apply calls will call both the diagram transactions and the equivalent reference model transaction. At the end of each transaction, the outputs for the MUT and the reference model are compared and logged to the simulation log file. Since the reference models have the same structure as the design, automatic code generators such as TestBencher can generate most of the reference model directly from the design.

Summary

C++ provides many benefits for modeling behavioral algorithms including advanced data structures, standard libraries and embedded software models. New hardware modeling libraries such as SystemC and TestBuilder make C++ useable for architectural modeling, but they are still difficult to use. Graphical code generation and GUI-based project management tools such as TestBencher Pro ease many of the problems incurred using C++ hardware modeling libraries. By choosing a combination of these new tools, users can automate the tedious aspects of model development, allowing them to focus on the functionality of their models.

Donna Mitchell is Vice President of Marketing and Co-Founder of SynaptiCAD Sales, Inc., 520 Prices Fork Rd., Blacksburg, VA 24060; (540) 953-3390; donna@syncad.com; www.syncad.com; www.syncad.com.