Integrate and Verify SystemC Models in a Graphical ESL Testbench

Transactional-level SystemC models can be easily integrated into a graphical ESL modeling and simulation environment by using simple wrappers.

An electronic-system-level (ESL) modeling and simulation environment allows the user to specify multi-processor systems using mixed graphical and textual notations. Transactional-level SystemC code is automatically generated from these descriptions. The system's behavior and performance can be analyzed early and rapidly by executing these abstract models in complete independence of the availability of embedded software code or hardware intellectual-property (IP) blocks.

More detailed SystemC models, such as those using GreenBus (provided as open source by the GreenSocs project), can be integrated into the ESL environment for verification purposes. In this case, the ESL modeling and simulation environment can serve to create and generate a system-level SystemC testbench. That testbench validates platform models. It also analyzes the impact of parameters on the behavior and performance of the platform model. Such parameters range from memory-access times to protocols for programmer’s view (PV) or programmer’s view timed (PVT) abstraction levels. This article describes how a message-level (or transaction-level-3) ESL modeling and simulation environment can be used as a testbench creation and generation environment. It also details how designers can rapidly integrate, analyze, and verify lower-level platform models (transaction levels 2 and 1), such as the ones found in the GreenSocs project.

Integration And Verification Principle

The integration of the external SystemC code into the ESL environment is done simply by importing header/source or object/library files. Those files will be compiled and linked to the overall SystemC executable model, including the testbench, which is automatically generated by the ESL environment from graphical and textual entries.

The graphical ESL test function, which is shown on the left-hand side in Figure 1 (Test_Function), represents the test case. While the ESL function is very simple for this example, the graphical-abstraction capability enables the rapid expression of complex and realistic use cases. A minimal 10X improvement in testbench-development productivity can be expected from graphical modeling compared to hand-coded SystemC.

This function periodically sends messages to a memory in the SystemC model. A message includes:

- A command (read/write)
- An address (indicating which memory is concerned by the command)
- Data (corresponding to the information that is sent/received to/from the memories)

The C code that defines this data structure in the ESL testbench is shown in Figure 2.

Figure 1: This graphic shows the integration and validation of a GreenSocs model into an ESL testbench.
typedef struct{
    int Address;
    const char* Data;
    typedefCommand Command;
}DefTokenWrapper;

Figure 2: The data-structure definition is shown here.

During each period—after sending a read and a write command—this function checks that the data received from the SystemC model matches the data sent to this model. This permits automatic validation of the behavior of the SystemC model in the ESL testbench. The duration of the period of the ESL testbench function is a parameter that’s used to stress the SystemC model by sending messages with a short period (high frequency). This function also can send commands to write data according to the GreenSocs PV or PVT protocols in the model. The protocol selection depends on a generic parameter set in the ESL testbench.

The wrapper function, which is shown on the right-hand side in Figure 1, acts as a master to the SystemC model. It unpacks incoming messages that are sent from the ESL function through a message queue (ToWrapper) and converts them to GreenSocs-specific PV/PVT transactions (e.g., to Transact(), Request.block(), SendData() primitives, etc.) with the router (init/target relation). The wrapper also packs data that come from the router in order to send messages back to the ESL test function through another message queue (FromWrapper).

Instead of graphical notations, the behavior of the wrapper function is described using simple SystemC code. Figure 3 details the code that implements the behavior of the wrapper in a SystemC thread (run). The conversion of a message to a PV/PVT transaction requires only a few lines of code.

Figure 4 shows the impact of the generic parameter that sets the PV/PVT protocol on the time it takes to write data in a memory and read data from that same memory. The state of the ESL testbench function (Test_Function) and SystemC model (External_SystemC_Model) is represented horizontally. An element is active when its state is represented with a plain red line. The element is inactive when it is represented with a dotted yellow line. Note that the Test_Function and External_SystemC_Model can be active in parallel. Communication between the ESL test function and SystemC model through message queues is represented using vertical arrows.

The timed-execution and message-sequence diagram shown in Figure 4 is automatically generated by the ESL modeling and simulation environment. It’s obtained through the interpretation of traces resulting from the execution of the automatically generated and instrumented SystemC model. That model includes the testbench, wrapper, and SystemC model.

The impact of the PV protocol is shown above the impact of the PVT protocol. The results show that it takes only 70 ns to write and read data to/from a memory when using PV. In contrast, it takes 440 ns when using PVT in our example. This result stems from the fact that a PV write primitive doesn’t utilize the GreenBus and its scheduler. Yet a blocking PVT write primitive does use them.
These results are obtained within minutes in the ESL verification environment used for this example. They enable the designers to determine optimal parameter values. For example, the optimal parameter value for the period of the ESL test function is approximately 600 ns when using PVT and 200 ns when using PV. This indicates that the execution of read/write commands in the SystemC model can be validated for all memories from the ESL test function in 800 ns when using the PV (see Figure 5).

Transitional-level SystemC models can be easily integrated into a message-level, graphical ESL modeling and simulation environment through the use of simple wrappers. Those wrappers convert generic transactions in the graphical ESL models to more specific PV/PVT transactions in the SystemC models. For the open-source GreenSocs-model example used in this article, the verification and validation of the models’ behavior was automated by using CoFluent Studio as a testbench creation and generation environment. It provided the textual and graphical model creation, automatic SystemC generation, and advanced analysis capabilities. Modeling a testbench using efficient graphical notations, which represent the parallel execution of functions performing computation and communication actions, provides a significant increase in the capacity for representing application-oriented complex use cases and in modeling productivity (a 10X improvement compared to hand coding). Moreover, optimal test configurations can be found within minutes by analyzing the impact of generic parameters.

Designers can analyze the impact of generic parameters on the behavior and performance of the SystemC model without having to change or recompile the model for each parameter value. Figure 7 shows the impact of the generic parameter, MemoryDelay_ns, on the memory read time when using PV. When the value of MemoryDelay_ns is set to 10 ns, it requires 190 ns to validate the behavior of a memory from Test_Function. When this parameter value is set to 40 ns, the same task requires 220 ns.

Transactional-level SystemC models can be easily integrated into a message-level, graphical ESL modeling and simulation environment through the use of simple wrappers. Those wrappers convert generic transactions in the graphical ESL models to more specific PV/PVT transactions in the SystemC models. For the open-source GreenSocs-model example used in this article, the verification and validation of the models’ behavior was automated by using CoFluent Studio as a testbench creation and generation environment. It provided the textual and graphical model creation, automatic SystemC generation, and advanced analysis capabilities. Modeling a testbench using efficient graphical notations, which represent the parallel execution of functions performing computation and communication actions, provides a significant increase in the capacity for representing application-oriented complex use cases and in modeling productivity (a 10X improvement compared to hand coding). Moreover, optimal test configurations can be found within minutes by analyzing the impact of generic parameters.

Jerôme Lemaitre completed a post graduate degree in 2002 at ESPEO, University of Orléans, France, majoring in embedded systems. From 2002 to 2007, he worked as a signal processing designer for the Dutch Foundation for research in Radio Astronomy. He is now solution specialist at CoFluent Design. His fields of interest include architecture exploration and performance analysis, MPSoC/NoC design and rapid prototyping.

Dr Mark Burton is the founder of GreenSocs. Mark graduated from Warwick university with a degree in computer systems engineering. He worked for Inmos and then moved to ACRI (The Advanced Computer Research Institute). He completed a PhD in Artificial Intelligence within Education, specifically focusing on the simulation of high level collaborative learning processes. He worked for ARM and is currently the chair of the OCP-IP SLD WG.