

ESL Synthesis Extensions (ESE) for SystemC are aimed at correcting the main deficiencies in SystemC for modeling, hardware-accuracy and synthesis. ESE simplifies the modeling of complex concurrency over threads and improves model composition through automated, formal interface contracts. While ESE elevates the description of control and complex datapaths, Bluespec's synthesis technology converts ESE designs into efficient RTL.

ESE is meant to be used by SystemC-based architects, hardware designers and verification engineers responsible for a chip's architecture and modeling. For hardware designers, it offers a higher level of design for control and complex datapaths. Unlike current approaches, it provides a seamless path from architecture exploration to design and verification. It reduces verification cycles and enables the synthesis to RTL code in Verilog with high quality of results (QoR). This Verilog code serves as input to any standard industry flow including RTL synthesis and simulation.

For architects, ESE provides a single environment for architecture exploration and design implementation, eliminating the need to create separate designs for modeling and implementation. It enables more precise concurrency modeling and introduces the ability to accurately assess the implications of different architectures for power, area, latency and throughput.

For verification engineers, it simplifies the expression of complex, multiple concurrent activities, while retaining all the other benefits of SystemC for verification. It accelerates the development of testbenches and offers the option of synthesizing testbenches for hardware emulation.

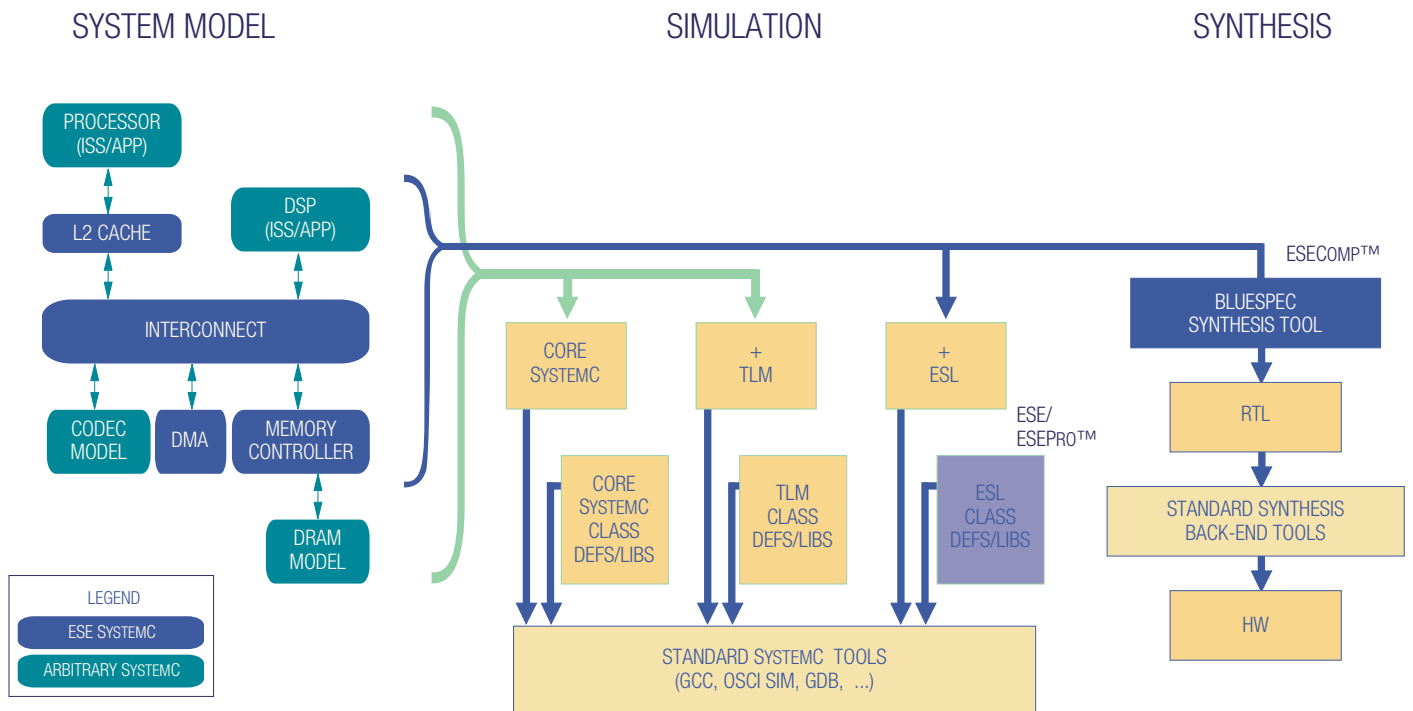
“Bluespec is offering the first ESL control logic synthesizer”
 - Gary Smith, chief EDA analyst at Gartner Dataquest

The Deficiencies of SystemC

Modeling Deficiencies: The concurrency model in SystemC consists of threads synchronized by events. Recent experience with SystemC, as well as decades of experience in the field of parallel software engineering (POSIX threads, semaphores, locks, etc.) have shown this concurrency model to be very weak for modeling complex concurrency, i.e., situations that have tens or more of threads with dynamic access patterns to hundreds or more of fine-grain, shared resources. Explicitly managing access to shared resources via events and locks in such situations is highly error-prone, leading to race conditions, deadlocks, etc. Such high degrees of concurrency also substantially reduce simulation speed. We are faced with just such situations when modeling complex hardware systems.

Hardware-Accuracy Deficiencies: For hardware accuracy, one needs both hardware-accurate data types (such as bit vectors of various widths) as well as hardware-accurate ways of expressing behavior and communication. SystemC has good solutions for the former but not the latter. The computation model of C or C++ threads is so far removed from hardware models (sequential statement execution, globally accessible shared memory with constant access time, global variables, recursion, stacks, heaps, ...) that it is hard for anyone to imagine exactly what hardware

ESL Synthesis Extensions (ESE) to SystemC



structures they are describing when writing such code. A good computational model needs to account for the static structure of hardware, the fine-grained parallelism in hardware, the communication costs of moving data between hardware structures, the resource costs of hardware operations, the costs of sharing hardware resources, and ultimately the constraints of clocked synchronous logic.

Synthesis Deficiencies: Perhaps as a natural consequence of the distance to hardware semantics described above, there is no good solution for synthesizing acceptable hardware from arbitrary SystemC models. In certain limited scenarios (e.g., loop-and-array computations as found in many DSP algorithms), good synthesis may be possible ("behavioral synthesis"), but for the vast majority of hardware structures, involving more complex control and communication, good synthesis requires a different solution.

ESE Solutions

ESL Synthesis Extensions for SystemC address all the deficiencies described above.

In software engineering of complex concurrent systems (operating systems, large-scale parallel database, transaction-processing systems and distributed systems), researchers and practitioners long ago formalized the concept of *atomic transactions* to manage complexity. An atomic transaction expresses a potentially complex update to a set of shared resources, with the guarantee that (a) either the whole update happens or none, and (b) no other atomic transaction (concurrent activity) can observe or affect the state during such a transaction. These properties form a powerful basis for specifying and describing *correct* behavior: the correctness of the entire system is implied by the correctness of individual transactions. So, instead of worrying about all the potential interactions between all the concurrent activities in the system, the programmer/designer can concentrate on individual atomic transactions.

The ESL Synthesis Extensions provide exactly this capability and benefit. System behavior is described as a collection of *rules*, where each rule is an atomic transaction consisting of a *condition* (guarding a complex update) and a set of *actions* (the complex update). Programmers find it far easier to express complex concurrent behaviors in this way, compared to working with threads and events.

Rule-based Interface Methods extend SystemC's interface mechanisms, providing a way to *modularize* or *object-orient* rules, i.e., to compose rules from fragments in different modules. A rule in module A can invoke *interface methods* (rule fragments) in other modules B, C, ... An interface method of a module B is a fragment of a rule containing a condition and an action that is *local* to the module B. The composite rule continues to have atomic semantics, and the benefits thereof.

Rules and Rule-based Interface Methods are orthogonal to levels of abstraction. They can be used both at very high-level, coarse-grained, transaction levels, as well as at low-level, fine-grained hardware-detailed levels.

How do Rules and Rule-based interface methods address the deficiencies of SystemC described above?

- 1) The atomic transaction semantics of Rules and Rule-based Interface Methods dramatically improve the expressive power of the language for specifying systems with complex concurrency, and they improve the inherent correctness of such specifications.
- 2) Rule-based Interface Methods are naturally *transactional* while having a perfectly meaningful hardware reading. In SystemC, one often uses TLM (Transaction Level Modeling) interfaces in the early phases of description, and then rewrites these into SC_SIGNAL-based interfaces when refining it into a hardware description. No such jump in semantics is needed when using Rule-based Interface methods - a single paradigm may be used all the way from high-level modeling down to detailed hardware description.

- 3) Rule has a natural reading as a State Transition in a Finite State Machine, i.e., rules do not have the chasm to hardware semantics that threads and events have. It is quite easy for the designer to think in hardware terms, to understand the hardware structures implied by a set of rules.
- 4) Rule-based Interface Methods naturally capture the idea of *communication* between modules, i.e., they appeal both to the object-oriented software view as well as to the hardware-oriented interface communication view.
- 5) Finally, Bluespec has more than seven years of experience in *synthesizing* system descriptions based on Rules and Rule-based Interface Methods into *high-quality hardware*, i.e., hardware that is comparable to hand-coded RTL in area, speed, power, etc., and that is produced and verified in half the time, or less, because of the expressive power of Rules.

In summary, Rules and Rule-based Interface Methods bridge the existing gap in SystemC between high-level modeling and detailed hardware design. Further, even detailed hardware designs, when written using Rules, are at a much higher level of abstraction than SC_SIGNAL-level SystemC (or RTL). A single paradigm can now be used to span that vertical spectrum of abstraction levels.

Bluespec SystemC Products

Bluespec's SystemC products support Linux operating systems:

ESE

This open and free implementation of the ESL Synthesis Extensions to SystemC is available via Bluespec's website. This version supports the ESL Synthesis Extensions for untimed simulations with OSCI or commercial SystemC simulators.

ESEPro

This is Bluespec's premium implementation of the ESL Synthesis Extensions to SystemC. This version adds support for clock-scheduled simulations enabling both:

- Untimed simulations with OSCI or commercial SystemC simulators; and,
- Clock-Scheduled simulations with OSCI or commercial SystemC simulators.

ESEComp

ESEComp synthesizes SystemC designs into highly efficient Verilog RTL.

Unique Core Technology: Term Rewriting Systems

Bluespec's patented technology is based on over seven years of commercial development and nine years of research at MIT. At the core of Bluespec's compiler technology is the introduction of the application of Term Rewriting Systems (TRS) to hardware synthesis. Term Rewriting Systems are a well-understood formalism from computer science. A TRS consists of "terms" which describe hardware states, and "rules" which describe behavior. A "rule" captures both a state change (an "action") and the conditions under which it can occur. As the rules in a Term Rewriting System have atomic semantics, analysis of hardware can be done even though it may be highly concurrent and complex. Commercial projects based on this technology have shown a reduction in DV time ranging from 50% to 70% immediately following initial training.

Further information can be read on www.bluespec.com about additional technical solution areas such as verification time reduction and low power optimization.

Bluespec's three day training course gives designers the tools they need to be successful and show material productivity improvement on their first project. Please contact your sales representative to sign up.