Chapter 10

EPILOGUE

In this epilogue, we reemphasize the importance of creating multi-MoC extensions of SystemC one more time, and summarize some of the contributions and goals of this book. The basic objective of this endeavor has been to disseminate our experience in extending SystemC with a multi-MoC kernel implementation, and the implementation itself.

One of the major advantages of SystemC over other forms of hardware description languages is the full expressive power of C++ at the designers disposal while modeling in SystemC. However, this advantage often shows the flip side of the coin, becoming a disadvantage. The freedom of using any C++ construct introduces programming errors, and often leads designers to use C++ constructs that are not synthesizable as hardware. Moreover, the lack of structure for creating models for specific MoCs leads to lack of *fidelity*. The term *fidelity* of a modeling framework in this case refers to an informal measure of how accurately one can model behaviors specific to an MoC using the modeling structures and facilities available in the framework. Besides fidelity, not exploiting the MoC specific properties of models implies less efficient simulation, as we have shown in the case of SDF models.

When SystemC was introduced in September 1999, there were two main selling points discussed in the industry and academia. First, SystemC is a class library based on C++, and hence any standard C++ compiler can create executables from SystemC models, which implies free simulation platform rather than expensive VHDL or Verilog simulator. Second, the flexibility of C++ allows designers to be creative, and using C++ makes it easier for software/hardware co-simulation, avoiding PLIs which incur lots of overhead during simulation.

It was quickly realized by industry practitioners that (i) using C++ by itself is not going to provide faster simulation, and (ii) free use of C++ is more of a liability than advantage, as synthesizability becomes an issue with arbitrary C++ constructs. In fact, if the level of abstraction remains at the RTL level, using C++, or using the industry best HDL simulators provide almost equivalent performance. However, if one has to synthesize hardware from SystemC model, it is almost necessary to remain at the RTL level of abstraction barring a few exceptions.

In 2001, SystemC-2.0 was introduced with some radical new features, and it borrowed a lot of concepts from the SpecC language. The most notable of those were the idea of channels, events, and interfaces. The idea of communication refinement, transaction level modeling, and interface based design were motivating factors for such changes. However, transaction level models are difficult to synthesize from, and tools that are commercialized since then can synthesize efficient hardware only from very limited set of constructs. Most problematic with such evolution of SystemC has been that heterogeneous and multi-MoC modeling does not have a direct support in SystemC-2.x yet.

Although we have discussed this extensively throughout this book, we would discuss this again here. Current SystemC simulation kernel is geared towards Discrete-Event (DE) simulation semantics, incorporating delta cycles which is very appropriate to model RTL level digital hardware, but not suitable for other Models of Computation. One can model any other Model of Computation by programmatic innovations, but eventually the simulation targeted kernel has the DE simulation semantics. This kernel uses dynamic event scheduling, and delta events to trigger delta cycles. For models which naturally belong to other alternative MoCs and are amenable to static scheduling, or other kinds of optimizations, when mapped to a DE kernel become inefficient in their simulation timing. So we believe that the only way SystemC can be made a full fledged system level design language is to enable SystemC to handle multi-MoC modeling and simulation, and support for behavioral hierarchy. This will allow designers to model systems which consist of heterogeneous components, modeled in different MoCs, and are hierarchically described. Moreover, the simulation of such models should not require flattening of hierarchy.

The reason why such heterogeneous modeling and simulation is important becomes clear if one looks at any embedded system or a System-on-Chip that goes in a consumer electronics equipment today. For example, a digital camera chip would consist of DSP, microcontrollers, A/D and D/A converters, memory elements and so on. Such systems are conglomerates of components best modeled in multiple MoC domains.

Embedded software or real-time light-weight operating system stacks could also be modeled in a heterogeneous modeling framework, which is currently difficult with SystemC.

SystemC standards body, and open SystemC initiative (OSCI) have been working over the last few years to make changes to SystemC standards to accommodate some of these needs. SystemC-AMS or SystemC-4.0 is slated to incorporate the libraries that will allow continuous domain modeling, which will facilitate the modeling of Analog and Mixed Signal Components. We are also aware of some activities related to software APIs for modeling embedded software in SystemC-3.0, but we are not aware of the current status of these efforts.

However, adding more libraries is not necessarily the solution to the problem at hand. Our belief is that once we create ways to adjoin multiple MoC specific simulation kernels and modeling constructs to SystemC, we will not only enable heterogeneous modeling, but also enable designers of SystemC based tools to easily add capabilities that are planned for SystemC-3.0 or SystemC-4.0.

We have therefore gone ahead, and created our prototype for extensions of SystemC-2.0.1 that enables us to create multi-MoC models, and allows us to exploit through the features of these enhanced kernels the MoC specific properties of these models to obtain simulation efficiency. In particular, in this book we have presented three MoC specific kernels, SDF, CSP and FSM, which we thought were very important MoCs for many embedded system components. Since our effort is limited by personnel and funding, we have not been able to provide a full industrial strength system, but we are putting forth a prototype-scale proof of concept. We have implemented three kernels, created some APIs that will allow others to add their own MoC specific kernels and functionalities, and we have created some heterogeneous models that use these kernels in conjunction. We have also shown efficiency gain in case of SDF, but due to lack of resources we have not done benchmarking for the CSP or FSM kernels, but we believe that with proper experimentation it would be easily revealed that exploiting MoC specific properties can only enhance the simulation performance.

Our hope is that this book would be able to convince some industry groups that multi-MoC extensions of SystemC is not only justified, it is necessary for SystemC to become a true system level modeling language. If our prototype can spark discussions within the SystemC community regarding the usefulness of such extensions, and about the best ways to implement such extensions, we would feel that our endeavor has been amply rewarded. Our implementation specifics of the design of the kernels may not be the only way or the best possible way to achieve these extensions, but it is one of the many possibilities. We urge the readers of the book to download our prototype, experiment with it, and send us comments and feedback [36].