Chapter 1

INTRODUCTION

1. Motivation

The technological advances experienced in the last decade has initiated an increasing trend towards IP-integration based System-on-Chip (SoC) design. Much of the driving force for these technological advances is the increasing miniaturization of integrated circuits. Due to the economics of these technologies where electronic components are fabricated onto a piece of silicon, the cost and size have exponentially reduced while increasing the performance of these components. Dating back to the early 1960s where two-transistor chips depicted leading edge technology, design engineers might have had difficulty envisioning the 100 million transistor chip mark that was achieved in 2001 [7]. Moore's law indicates that this progress is expected to continue up to twenty more years from today [44]. The ability to miniaturize systems onto a single chip promotes the design of more complex and often heterogeneous systems in both form and function [12]. For example, systems are becoming increasingly dependent on the concept of hardware and software co-design where hardware and software components are designed together such that once the hardware is fabricated onto a piece of silicon, the co-designed software is guaranteed to work correctly. On the other hand, a single chip can also perform multiple functions, for example network processors handle regular processor functionality as well as network protocol functionality. Consequently, the gap between the product complexity and the engineering effort needed for its realization is increasing drastically, commonly known as the *productivity gap*, as shown in Figure 1. Many factors contribute to the productivity gap

such as the lack of design methodologies, modeling frameworks & tools, hardware & software co-design environments and so on. Due to the hindering productivity gap, industries experience an unaffordable increase in design time making it difficult to meet the time-to-market. In our efforts to manage complex designs while reducing the productivity gap, we specifically address modeling and simulation frameworks that are generally crucial in the design cycle. In particular, we attempt at raising the level of abstraction to achieve closure of this productivity gap. In this effort, we build on the modeling and simulation framework of SystemC [49].

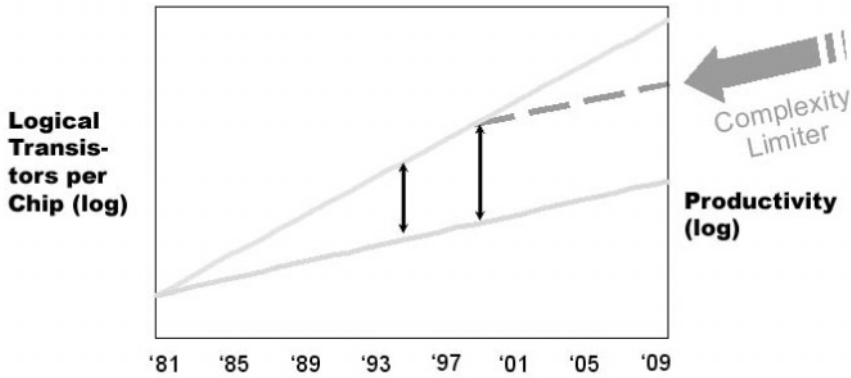

Figure 1.1. Productivity Gap [16]

2. System Level Design Languages and Frameworks

System Level Design Languages (SLDL) provide users with a collection of libraries of data types, kernels, and components accessible through either graphical or programmatic means to model systems and simulate the system behavior. A modeling framework provides a way to create a representation of models for simulation, and a simulation framework is a collection of core libraries that simulates a model.

In the past few years, we have seen the introduction of many SLDLs and frameworks such as SystemC, SpecC, SystemVerilog [49, 63, 65] etc., to manage the issue of complex designs. There also is an effort to lift the abstraction level in the hardware description languages, exemplified in the efforts to standardize SystemVerilog [65]. Another Java-based

system modeling framework developed by U. C. Berkeley is Ptolemy II [25]. The success of any of these languages/frameworks is predicated upon their successful adoption as a standard design entry language by the industry and the resulting closure of the *productivity and verification gaps*.

SystemC [49] is poised as one of the strong contenders for such a language. SystemC is an open-source SLDL that has a C++ based modeling environment. SystemC's advantage is that it has free simulation libraries to support the modeling framework. This provides designers with the ability to construct models of their systems and simulate them in a very VHDL and Verilog [69, 68] like manner. SystemC's framework is based on a simulation kernel which is essentially a scheduler. This kernel is responsible for simulating the model with a particular behavior. The current SystemC has a Discrete-Event based kernel that is very similar to the event-based VHDL simulator. However, the strive to attain a higher level of abstraction for closure of the productivity gap, while keeping the Discrete-Event (DE) simulation semantics are highly contradictory goals. In fact, most system models for SoCs are heterogeneous in nature, and encompass multiple Models of Computation (**MoC**) [37, 23] in its different components.

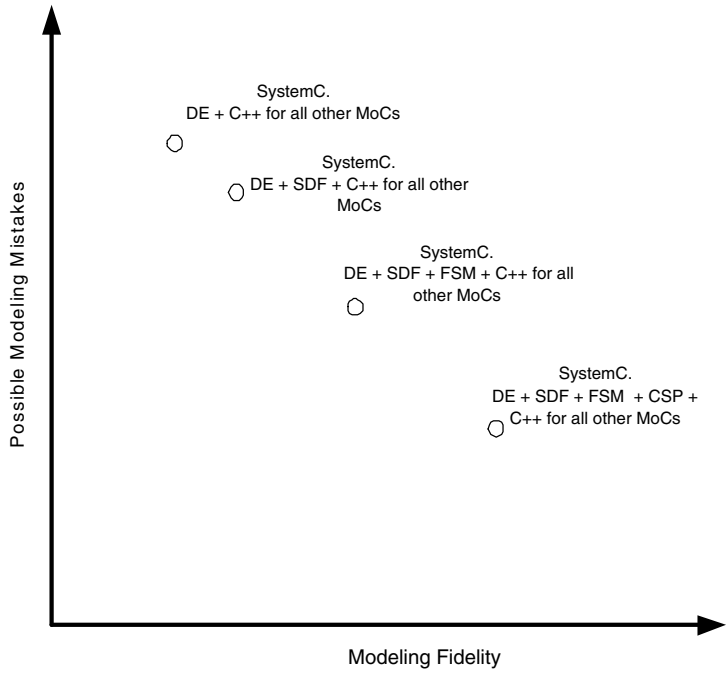

Figure 1.2. Possible Mistakes occurred versus Modeling Fidelity

Figure 1.2 displays the difficulty in being able to express MoC specific behaviors versus the possible modeling errors made by designers . Users restricted to DE semantics of SystemC lack in facilities to model other MoCs distinct from DE, requiring user-level manipulations for correct behavior of these MoCs. Due to these manipulations, designers may suffer from an increased number of modeling errors to achieve correct operation of a model. Hence, Figure 1.2 shows that SystemC with its DE kernel and functionality of C++ provides the most modeling expressiveness but also indicates more possible modeling mistakes. Conversely, the modeling expressiveness is significantly reduced when employing MoC-specific kernels for SystemC, but the possible errors made are also reduced.

We have therefore chosen the term 'fidelity' as a qualifying attribute for modeling frameworks. Restricting SystemC users to use only implemented MoC-specific structures and styles disallows users to use any feature afforded by free usage of C++, but such restricted framework offers higher fidelity. Fidelity here refers to the capability of the framework to faithfully model a theoretical MoC. It is necessary to think about simulation semantics of such a language away from the semantics of pure Discrete-Event based hardware models and instead the semantics should provide a way to express and simulate other Models of Computation. The inspiration of such a system is drawn upon the success of the Ptolemy II framework in specification and simulation of heterogeneous embedded software systems [25]. However, since SystemC is targeted to be the language of choice for semiconductor manufacturers as well as system designers, as opposed to Ptolemy II, its goals are more ambitious. Our focus is in developing an extension to the SystemC kernel to propose our extended SystemC as a possible heterogeneous SLDL. We demonstrate the approach by showing language extensions to SystemC for Synchronous Data Flow (SDF), Communicating Sequential Processes (CSP) and Finite State Machine (FSM) MoCs. A common use of SDF is in Digital Signal Processing (DSP) applications that require stream-based data models. CSP is used for exploring concurrency in models, and FSMs are used for controllers in hardware designs. Besides the primary objective of providing designers with a better structure in expressing these MoCs, the secondary objective is to gain simulation efficiency through modeling systems natural to their behavior.

2.1 Simulation Kernel and MoC

The responsibility of a simulation model is to capture the behavior of a system being modeled and the simulation kernel's responsibility is to simulate the correct behavior for that system. In essence, the Model of

Computation dictates the behavior which is realized by the simulation kernel. An MoC is a description mechanism that defines a set of rules to mimic a particular behavior [39, 37]. The described behavior is selected based on how suitable it is for the system. *Most MoCs describe how computation proceeds and the manner in which information is transferred between other communicating components as well as with other MoCs.* For more detailed discussion on MoCs, readers are referred to the preface of this book and [32, 25].

2.2 System Level Modeling and Simulation

For most design methodologies, simulation is the foremost stage in the validation cycle of a system, where designers can check for functional correctness, sufficient and appropriate communication interactions, and overall correctness of the conceptualization of the system. This stage is very important from which the entire product idea or concept is modeled and a working prototype of the system is produced. Not only does this suggest that the concept can be realized, but the process of designing simulation models also refines the design. Two major issues that need consideration when selecting an SLDL for modeling and simulation are *modeling fidelity* and *simulation efficiency*, respectively.

An SLDL's modeling fidelity refers to the constructs, language and design guidelines that facilitate in completely describing a system with various parts of the system expressed with the most appropriate MoCs. Some SLDLs make it difficult to construct particular models due to the lack of fidelity in the language. For example, SystemC is well suited for Discrete-Event based models but not necessarily for Synchronous Data Flow (SDF) models. [23, 45] suggest extra designer guidelines that are followed to construct Synchronous Data Flow models in SystemC. These guidelines are required because the SDF model is built with an underlying DE kernel, increasing the difficulty in expressing these types of models. This does not imply that designs other than DE-based systems cannot be modeled in SystemC, but indicates that expressing such models is more involved, requiring designer guidelines [23, 45].

Simulation efficiency is measured by the time taken to simulate a particular model. This is also a major issue because there are models that can take up to hours of execution time. For example, a PowerPC 750 architecture in SystemC [42] takes several hours to process certain testbenches [61]. With increased modeling fidelity through the addition of MoC kernels in SystemC, models for better simulation efficiency can be created.

Choosing the appropriate SLDL requires modeling frameworks to be appropriately *matched* to allow for meaningful designs. The simulation

framework also has to be *matched* to allow for correct and efficient validation of the system via simulation. The frameworks must provide sufficient behavior to represent the system under investigation and must also provide a natural modeling paradigm to ease designer involvement in construction of the model. Consequently, most industries develop proprietary modeling and simulation frameworks specific for their purpose, within which their product is modeled, simulated and validated. This makes simulation efficiency a gating factor in reducing the productivity gap. The proprietary solutions often lead to incompatibility when various parts of a model are reusable components purchased or imported as soft-IPs. Standardization of modeling languages and frameworks alleviate such interoperability issues.

To make the readers aware of the distinction between modeling guideline versus enforced modeling framework, we present a pictorial example of how SDF models are implemented in [23, 45] (Figure 1.3) using SystemC's DE kernel. This example shows a Finite Impulse Response (FIR) model with *sc_fifo* channels and each process is of *SC_THREAD()* type. This model employs the existing SystemC Discrete-Event kernel and requires handshaking between the Stimulus and FIR, and, FIR and Display blocks. The handshaking dictates which blocks execute, allowing the Stimulus block to prepare sufficient data for the FIR to perform its calculation followed by a handshake communication with the Display block.

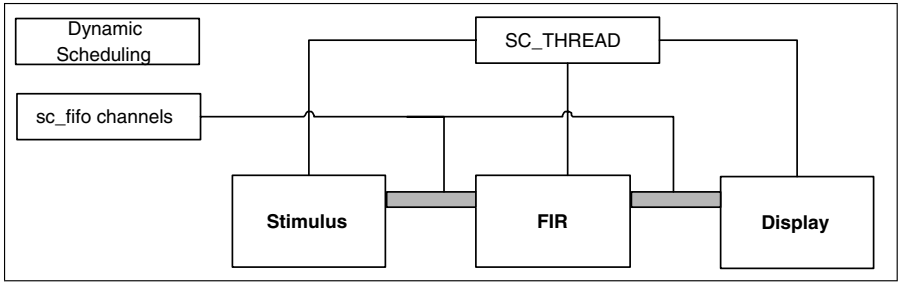

Figure 1.3. FIR model with Designer Guidelines [45, 23]

The same example with our SDF kernel is shown in Figure 1.4. This model uses *SDFport*s specifically created for this MoC for data passing instead of *sc_fifo*s and no synchronization is required since *static scheduling* is used. The model with the SDF kernel abandons any need for handshaking communication between the blocks and uses a scheduling algorithm at initialization time to determine the *execution schedule* of the blocks.

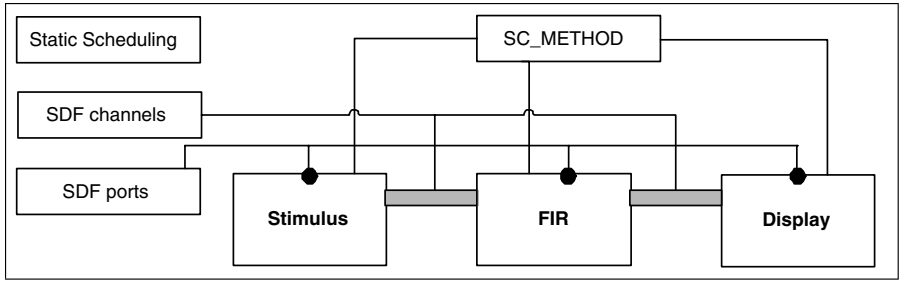

Figure 1.4. FIR model with our SDF kernel

We provide further explanation about these modeling examples in Chapters 3 and 5.

2.3 Simulation Efficiency Versus Fidelity of MoCs

It is conceivable that there can be large, detailed models that take enormous amounts of time for simulation [61]. We measure simulation performance in terms of the amount of time taken to simulate a model from start to end. We believe that matching components to models with the most appropriate MoC enhances simulation efficiency.

An MoC describes the underlying behavior of a system on top of which a model following that particular behavior is simulated. The systems expected behavior must be modeled appropriately by the modeling framework, otherwise an unnatural modeling scheme has to be used to impose the behavior and the simulation efficiency will almost always suffer. When we describe a simulation framework supporting the Model of Computation, we particularly refer to kernel-support for that MoC. Our motivation involves modeling and simulating SDF, CSP, FSM and other MoC designs in SystemC by introducing distinct simulation kernels for each MoC. Although meant to be a system level language, the current SystemC simulation kernel uses only a non-deterministic DE [39, 37, 49] kernel, which does not explicitly simulate SDF, CSP or FSM models in the way that they could be simulated more efficiently and easily.

In other words, current SystemC is very well suited for designs at the register transfer level of abstraction, similar to VHDL or Verilog models. However, since SystemC is also planned as a language at higher levels of abstraction, notably at the system level, such Discrete-Event based simulation is not the most efficient. An SoC design might consist of DSP cores, microprocessor models, bus models etc. Each of these may be most naturally expressible in their appropriate MoCs. To simulate such a model the kernel must support each of the MoCs interlaced in the

user model. This allows designers to put together heterogeneous models without worrying about the target simulation kernel and discovering methodologies to enforce other MoCs onto a single MoC kernel. Figure 1.5 shows an example of a digital analyzer [32] that employs two MoCs, one being an untimed Data Flow MoC and the other being a timed model based on the DE MoC. The Controller IP and Analyzer IP could be possibly obtained from different sources requiring the interfaces to allow for communication between the two. Support for modeling these IPs requires the simulation framework to provide the underlying MoCs, otherwise programmatic alterations are required to the original IPs for conformity. Again, it is crucial for modeling frameworks to allow designers to express their models in a simple and natural fashion. This can be achieved by implementing MoC-specific kernels for the modeling framework along with sufficient designer guidelines in constructing models.

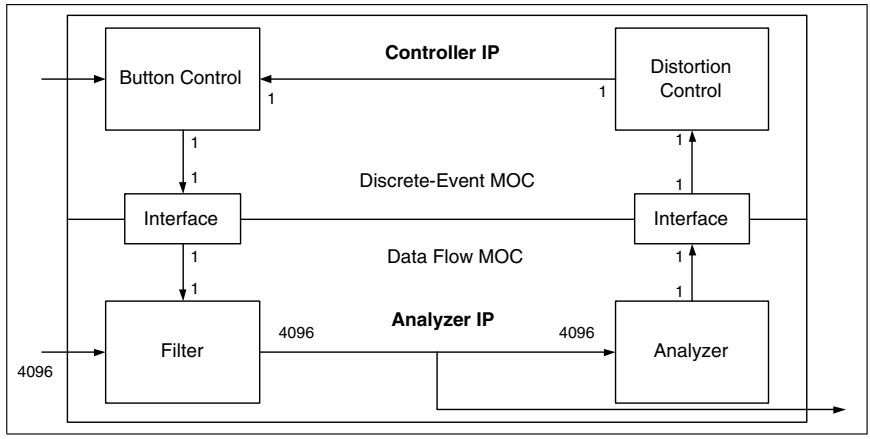

Figure 1.5. Example of Digital Analyzer using Models of Computation [32]

3. Our Approach to Heterogeneous Modeling in SystemC

Heterogeneous modeling in SystemC has been attempted before, but has only been attained by using designer guidelines [23, 45]. One of the first attempts at designer guidelines for heterogeneous modeling is provided in [23]. We find these guideline based approaches to be quite unsatisfactory. In [23] the authors model SDF systems using *SC_THREAD()* processes and blocking read and write *sc_fifo* channels. This scheme uses dynamic scheduling at runtime as opposed to possible static scheduling of SDF models. They accommodate an SDF model with the underlying

DE kernel. We argue that this is not an efficient method of implementing SDF in SystemC nor is it natural. The adder example they present [23] uses blocking *sc_fifo* type channels that generate events on every blocking read and write resulting in an increase in the number of delta cycles. Furthermore, the use of *SC_THREAD()* introduces context-switching overhead further reducing the simulation efficiency [20]. Although [23] promotes that tools may be designed to exploit static scheduling algorithms to make efficient SDF simulation in systems, no solution or change in the kernel was provided. They only hint at some source level transformation techniques in enabling SDF systems with the underlying DE reference implementation kernel.

Another effort in modeling Data Flow systems in SystemC was done in [45], where they provide an Orthogonal Frequency Division Multiplex (OFDM) example as an SDF model. However, this model does not exploit the SDF Model of Computation either since this example also uses blocking read and write resulting in dynamic scheduling. Once again this approach defeats the purpose of modeling SDF systems because once recognized as SDF, they should be statically scheduled. These models presented in [23, 45] can be converted such that there is no need for *SC_THREAD()* or *SC_CTHREAD()* processes, communication signals to pass control, and most importantly dynamic scheduling. We present an extension of SystemC in which an SDF kernel interoperable [14] with the existing DE kernel can eliminate the need for dynamic scheduling and designer guidelines specific for SDF models. Similar extensions are also prototyped for CSP and FSM Models of Computation.

3.1 Why SystemC?

In addition, the preset C++ language infrastructure and object- oriented nature of C++ used to define SystemC extends usability further than any existing hardware description language (HDL) at present. SystemC provides embedded system designers with a freely available modeling and simulation environment requiring only a C++ compiler. We have undertaken the task of building a SystemC simulation kernel that is interoperable with the existing simulation kernel of the reference implementation, but which employs different simulation kernel functions depending on which part of the design is being simulated. In this book, we focus on building the simulation kernel appropriate for the SDF [37, 39, 25], CSP [27] and FSM models.

Once kernel specific MoCs are developed, they can be used as standalone for a specific system and even in unison to build heterogeneous models. Communication between different kernels could either be event-driven or cycle-driven depending on the choice of the implementer. Event-

driven communication describes a system based on message passing in the form of events, where events trigger specific kernels to execute. As an example, our SDF kernel executes only when an event from the DE kernel is received. On the other hand cycle-driven refers to a system whereby for a number of cycles one particular kernel is executed followed by another.

Our goal is not to provide a complete solution, but to illustrate one way of extending SystemC. We hope this would convince the SystemC community to build industry strength tools that support heterogeneous modeling environments. However, the standardization of SystemC language is still underway, but as for the kernel specifications and implementations, SystemC only provides the Discrete-Event simulation kernel. We build upon the current SystemC standard in laying a foundation for an open source multi-domain modeling framework based on the C++ programming language which has the following advantages:

- The open-source nature of SystemC allows alterations to the original source.
- Object- oriented approach allows modular and object- oriented design of SystemC and its extensions to support the SDF kernel.
- C++ is a well accepted and commonly used language by industry and academics, hence making this implementation more valuable to users.

4. Main Contributions of this Book

We extend the existing SystemC modeling and simulation framework by implementing a number of MoC-specific kernels, such as SDF, CSP and FSM. Our kernels are interoperable with SystemC's DE kernel allowing designers to model and simulate heterogeneous models made from a collection of DE and SDF models or other combinations corresponding to different parts of a system. By extending the simulation kernel, we also improve the modeling fidelity of SystemC to naturally construct models for SDF, CSP and FSM. Furthermore, we increase the simulation efficiency of SDF and DE-SDF models in SystemC. We do not benchmark for the other kernels, but we believe they will also show similar improvements.

Synchronous Data Flow MoC is by no means a new idea on its own and extensive work has been done in that area. [5] shows work on scheduling SDFs, buffer size minimization and other code optimization techniques. We employ techniques presented in that work, such as their scheduling algorithm for SDFs. However, the focus of [5] pertains to

synthesis of embedded C code from SDF specifications. This allows their SDF representations to be abstract, expressing only the nodes, their connections, and token values. This is unlike the interaction we have to consider when investigating the simulation kernel for SystemC. We need to manipulate the kernel upon computing the schedules requiring us to discuss the Data Flow theory and scheduling mechanisms. Our aim is not in just providing evidence to show that changes made in the kernel to accommodate different MoCs improves simulation efficiency, instead, we aim at introducing a heterogeneous modeling and simulation framework in SystemC that does also result in simulation efficiency.

Hoare's [27] Communicating Sequential Processes is an MoC for modeling concurrent systems with a rendez-vous style synchronization between processes. Rendez-vous occurs between two processes ready to communicate. Otherwise, the process attempting the communication suspends itself and can only be resumed once the process receiving the transfer of data from that process is ready to communicate. We implement the CSP kernel with SystemC's coroutine packages used to create thread processes. CSP processes in our CSP kernel are SystemC thread processes with a different scheduling mechanism suitable for CSP modeling.

Most hardware systems have controllers modeled as FSMs. The current SystemC can model FSMs using standard switch statements, but the FSM kernel provides a direct method of invoking different states. Combining SDF, CSP and FSM MoCs to create heterogeneous models shows the usefulness of our multi-MoC SystemC framework. The famous Dining Philosophers problem has an elegant solution that can be modeled in CSP. Furthermore, a deadlock avoidance technique can be added using an FSM controller as a footman assigning seats. We discuss this model in further detail in Chapter 9.

Further improvements in synthesis tools such as Cynthesizer and Cocentric SystemC Compiler [17, 64] will provide designers with the capability of effective high level synthesis. We hope to initiate the process by which SystemC can fulfill its purpose of being a high level and multi-domain modeling framework for industry use. We plan on distributing our prototype implementation that introduces heterogeneity in SystemC to the SystemC community via a freely downloadable distribution [36].