ConcurrenC: A Novel Model of Computation for Effective Abstraction of C-based SLDLs

Weiwei Chen, Rainer Dömer

Technical Report CECS-09-07
May 27, 2009

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{weiweic, doemer}@uci.edu
http://www.cecs.uci.edu/

ConcurrenC: A Novel Model of Computation for Effective Abstraction of C-based SLDLs

Weiwei Chen, Rainer Dömer

Technical Report CECS-09-07
May 27, 2009

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{weiweic, doemer}@uci.edu
http://www.cecs.uci.edu

Abstract

System design in general can only be successful if it is based on a suitable formal Model of Computation (MoC) that can be well represented in an executable System-level Description Language (SLDL) and is supported by a matching set of design tools. While C-based SLDLs are popular in system-level modeling and validation, current tool flows impose almost arbitrary restrictions on the synthesizable subset of the supported SLDL. A properly aligned and consistent system-level MoC is often neglected or even ignored.

In this report, we motivate the need for a well-defined MoC in system design. We discuss the close relationship between SLDLs and the abstract models they can represent, in contrast to the smaller set of models the tools can support. Based on these findings, we then propose a novel MoC, called ConcurrenC*, that defines a clear system level of abstraction, aptly fits system modeling requirements, and can be expressed precisely in both SystemC and SpecC SLDLs. Using a H.264 video decoder example, we demonstrate how the proposed ConcurrenC MoC fits the features and characteristics of a real-world embedded application.*

This work has been supported in part by NSF Grant #0747523

Contents

ConcurrenC: A Novel Model of Computation for Effective Abstraction of C-based SLDLs

Weiwei Chen, Rainer Dömer

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

{weiweic, doemer}@uci.edu
http://www.cecs.uci.edu

Abstract

System design in general can only be successful if it is based on a suitable formal Model of Computation (MoC) that can be well represented in an executable System-level Description Language (SLDL) and is supported by a matching set of design tools. While C-based SLDLs are popular in system-level modeling and validation, current tool flows impose almost arbitrary restrictions on the synthesizable subset of the supported SLDL. A properly aligned and consistent system-level MoC is often neglected or even ignored.

In this report, we motivate the need for a well-defined MoC in system design. We discuss the close relationship between SLDLs and the abstract models they can represent, in contrast to the smaller set of models the tools can support. Based on these findings, we then propose a novel MoC, called ConcurrenC*, that defines a clear system level of abstraction, aptly fits system modeling requirements, and can be expressed precisely in both SystemC and SpecC SLDLs. Using a H.264 video decoder example, we demonstrate how the proposed ConcurrenC MoC fits the features and characteristics of a real-world embedded application.*

This work has been supported in part by NSF Grant #0747523

1 Introduction

Embedded system is one of the most popular computational systems in our current information era. With applications ranging from portable media player to medical equipments, from consumer electronic devices to communication satellites, from real-time automotive applications to highly-reliable space transportation systems, embedded system has a profound impact on our everyday life.

An embedded system is a special-purpose computer system which is designed to perform one or multiple functions. It usually has tight constraints on its size, power, timing, and cost. With the development of semiconductor technology, embedded systems gain a tremendous amount of functionality and processing ability by integrating multiple components / intellectual properties (IP) onto a single Multi-Processor System-on-Chip (MPSoC). The complexity of the system is growing rapidly with the increasing of those integrated components which have to cooperate properly and run in parallelism. Due to the complexity and tight constraints of the expected system, design engineers are facing great challenges to build up a satisfactory system within a short time-to-market design period.

According to the 2007 edition of the International Technology Roadmap for Semiconductors (ITRS) [13], system-level design, listed as one of the top overall design challenges for productivity, is a promising solution to improve the design productivity. Critical factors include high level of abstraction and platform-based design. Systems have been described at different abstraction levels (block diagram, state charts, program model, etc.) but with mismatching design automation tools for many years. A new matching abstract system model is needed to simplify design, including simulation, estimation, synthesis, verification, implementation, and design space exploration [13].

In this report, we aim to establish a properly aligned relation between the three essential ingredients for successful system design, namely (1) a suitable formal Model of Computation (MoC), (2) an executable System-level Description Language (SLDL), and (3) a matching set of design tools.

1.1 System-level Design

Figure 1 illustrates that the need of additional software required for hardware is doubling every ten months, and the capability of technology is doubling every thirty-six months. On the other hand, since the hardware productivity improved over the last several years by putting multiple cores on to a single chip, the productivity especially for hardware-dependent software is far behind which is doubling only every five years. Thus System-on-Chip design productivity cannot follow the speed of the nanoelectronics technology development which is characterized by Moore's Law. As a result, an additional higher level of abstraction - the so-called System Level - should be introduced and utilized.

Sangiovanni-Vincentelli mentioned in [23] that system-level design is placed as "a level above RTL including both HW and SW design". More specifically, System-level design is defined to "consist of a behavioral (before HW/SW partition) and architectural level (after)" and is claimed to increase productivity by 200K gates/designer year.

According to ITRS 2007, at system-level, silicon resources are defined in terms of abstract functions and blocks. Abstract functions are related with embedded software, e.g. embedded coded high-level and assembly programming language, configuration data, etc. And blocks are related with embedded hardware, e.g. processing cores, buses, peripherals, reconfigurable components, etc. Hardware (HW) corresponds to implemented electronic circuit components, while software

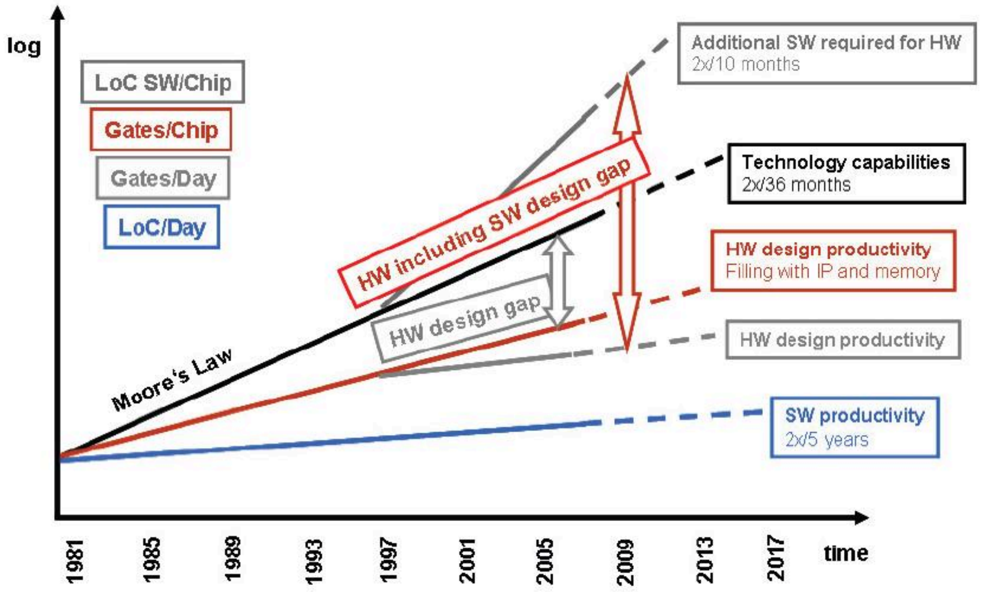


Figure 1: Hardware and software design gaps over time [13]

(SW) corresponds to those logic abstract functions performed on the hardware. A whole embedded system should be a well designed integration of hardware and software elements.

There are two independent degrees of design freedom, behavior and architecture, where the aggregate of behavior defines the function of a system and the aggregate of architecture defines the system platform. Platform mapping from system functionality onto platform architecture is the major task of embedded system-level design due to the special purpose and tight constraints of the embedded systems. Now, this task becomes more difficult with the increased system complexity and heterogeneity [13].

ITRS 2007 also mentioned that systems have been reasoned about at different abstraction levels (block diagram, state charts, program model, etc.) but with little support of design automation tools for many years. Such situation should be changed in the near future to achieve proper design productivity. New abstract system models are needed to simplify design tasks, including simulation, estimation, synthesis, verification, system implementation, and design space exploration [13].

Figure 2 predicts that the research for HW-SW co-design and verification will be required until 2010 and the development of such technology will be underway till 2014. Therefore, system-level design methodology research is a necessary, promising, and long-term goal for semiconductor industry.

1.2 Abstract Modeling

A model is the abstraction of reality. An embedded system model is the representation of actual and intended characteristics of the whole system. Only a well-defined model will reflect the essential

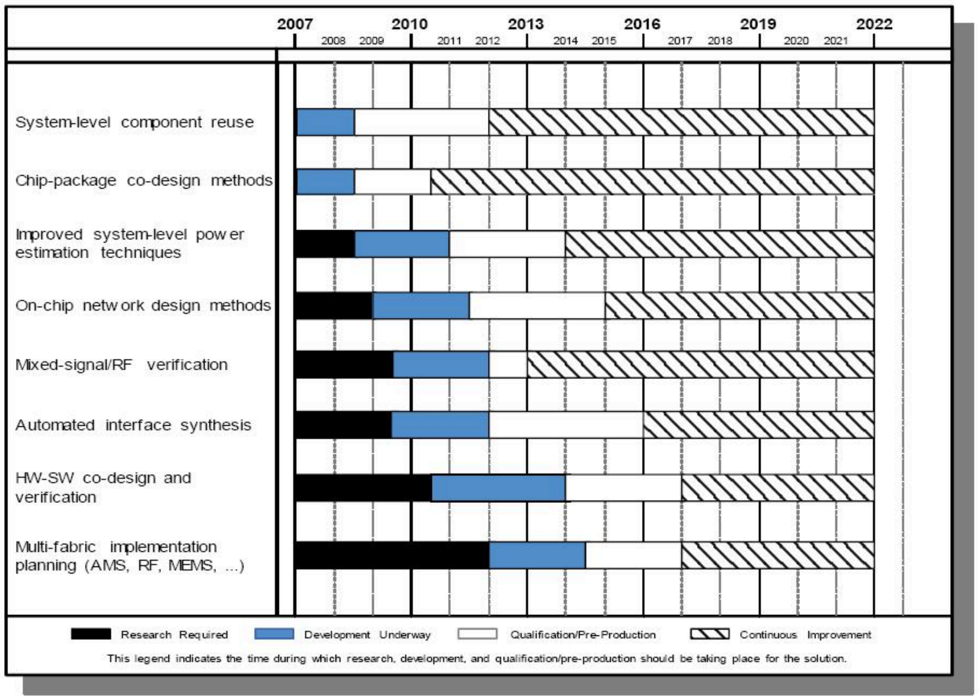


Figure 2: System-level design potential solutions [13]

features of the system and provides the possibility for efficient system design and implementation.

In system-level design, abstract modeling is of most importance. A proper abstraction and specification system model is key to accomplish efficient design tasks, including simulation, estimation, synthesis, verification, design space exploration, and the final successful implementation. The need of abstract models is significant since it is the foundation of the following design tasks. However, a lot of research tasks have focused on the steps after the system specification phase, such as synthesis and verification. Little has been done on modeling itself. The complexity of the system is an obstacle. Another challenge is that the quality of the model cannot be straightforward measured and compared.

The level of abstraction is an essential issue for modeling. A good model should reflect the essential functions and properties of the system and hide those implementation details which distract from the system-level view and increase the difficulty of the design tasks. Ideally, multiple well-defined abstraction levels are needed to enable gradual system refinement and synthesis by adding more details at each step. In other words, a good abstraction level should retain only those features desired by the current job, and abstract away the other unnecessary details.

2 Related Work

There has been considerable effort on the improvement of system design. Related work exists on many models of computation and SLDLs, as well as existing system design flows.

2.1 Model of Computation

Edwards et al. argue in [5] that the design approach should be based on the use of formal methods to describe the system behavior at a higher level of abstraction. A Model of Computation (MoC) is such formal method for system design. MoC is a formal definition of the set of allowable operations used in computation and their respective costs [17]. This defines the behavior of the system as at a certain abstract level to reflect the essential system features. Many different models of computation have been proposed for different domains. An overview can be found in [9] and [16].

2.1.1 Kahn Process Network

Kahn Process Network (KPN) is a deterministic MoC where deterministic processes are connected by unbounded FIFO communication channels [15]. The output of the KPN network does not depend on the execution order of the processes. Therefore, it is quite flexible to schedule a KPN network. For a KPN system, no global scheduler is needed and the partitioning over a number of KPN components is a simple task since the control is distributed to individual processes. Data exchanges are distributed over FIFO channels between processes and therefore resource contention is avoided. However, unbounded FIFOs are not realistic for real-world implementation. Artificial deadlock may occur if the FIFO bounds are not safely designed. Proofs have been provided that in general it is an undecidable problem to test whether a KPN is strictly bounded or not [20].

Dataflow Process Network (DFPN) [20], is a special case of KPN in which dataflow can be shown as a process network and the size of the communication buffers is bounded. A variety of dataflow models are proposed and studied over the years. Lee and Messerschmitt [3] discussed *Synchronous dataflow (SDF)* in which each actor consumes and produces a fixed number of tokens on each port of each firing and static scheduling is possible on a single processor and on multiple processors in parallel. Engels et al. [7] proposed *Cyclo-static dataflow* whose actors' consumption and production rates vary in a cyclic but predetermined pattern. *Heterochronous dataflow (HDF)* [11] by Girault and Lee composes finite state machine (FSM) with SDF, allowing actors to change their rate signatures between global iterations of a model. *Parameterized Synchronous dataflow (PSDF)* [1] discussed by Bhattacharya and Bhattacharyya is an extension of synchronous dataflow which is suitable for reconfiguration. Buck [14] proposed the *Boolean dataflow (BDF)* which allows the use of dynamic actors, like BooleanSelect and BooleanSwitch, and is Turing-complete so that the static analysis does not guarantee that the scheduling algorithm will always succeed. Zhou [27] discussed *Dynamic dataflow (DDF)* which adopts runtime analysis without answering the question of deadlock analysis and boundedness statically.

Software/hardware integration medium (SHIM) [6] is a concurrent asynchronous deterministic model which is essentially an effective KPN with rendezvous communication for heterogeneous embedded systems. According to Edwards et al. [6], concurrency, asynchronicity and determinism are the three major characteristics of the SHIM model. Concurrency models the cooperating work among those hardware components and software modules. Asynchronicity allows the abstraction of the software details in order to deal with the difficulty in predicting the software timing issues due to the complexity in instruction interaction at the low level architecture, like cache, pipeline, branch predictions. Determinism ensures the behavior of the simulator, simplifies debugging due to the bug reproduction, and provides benefits for formal verification by reducing the number of the possible behaviors and computational burden.

While these MoCs are popular for modeling signal processing applications, they are not well-suited for controller applications.

2.1.2 Petri Net

Petri Net, an abstract, formal model of information flow, is a state-oriented hierarchical model, especially for systems that are concurrent, distributed, asynchronous, parallel, non-deterministic, or stochastic activities [21]. The Petri net model consists of a set of places, a set of transitions, and a set of tokens. Tokens reside in places, and circulate through the whole net, being consumed and produced according to transition fires. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems by using Petri Net [18]. Petri nets have a great ability to model different systems, like computer hardware, computer software, concurrent systems. They also have the ability to describe a system hierarchically by replacing an entire net by a single place at a more abstract level, or a transition, or represent places and transitions by subnets to provide more detail refinement. However, Petri nets are uninterpreted models which means in general no meaning is attached to the places and transitions in the net and it deals only with the abstract features inherent in the structure of the net. Petri net can also quickly become incomprehensible with a system complexity increase.

2.1.3 Dataflow Graph and Finite State Machine

Dataflow Graph (DFG) and its derivatives are MoCs for describing computational intensive systems [10]. DFG is a directed graph consisting of nodes where the nodes represent operations or functions, and the arcs represent data flow through the graph. DFG model has two principles: asynchrony and functionality. The asynchrony principle states that all operations cannot be executed until the required operands are available. This implies the data dependency of the system. The functionality principle states that all operations behave as functions which do not have any side effects. It means that any two enabled operations can be executed in either order, or in parallel. DFG is excellent for representing computations described by complex functions. It is very popular for describing DSP components and systems. However, DFG is not suitable to represent control parts which are commonly found in most programming languages.

Finite State Machine (FSM) is popular for describing control systems and is widely adopted in hardware design since the temporal behavior of the system can naturally be represented in the form of states and transitions [10]. The FSM model consists of a set of states, a set of transitions between states, and a set of actions associated with these states or transitions. FSM is simple and easy for design, implementation and execution. Given a set of inputs and a known current state, the state transitions can be predicted and thus allow easy testing. FSM also helps to transfer from a meaningful abstract representation to a specific implementation. It is flexible to implement a FSM-based system in different topology, and it is easy to incorporate many other techniques. FMS also has some limitations. The conditions for state transitions are fixed and cannot be dynamically changed. Larger systems implemented by using FSM can be difficult to manage and maintain since the model becomes incomprehensible when the complexity of the system increases. Furthermore, FSM should only be used when a systems behavior can be decomposed into separate states with well defined conditions for state transitions. It is not easy to clearly separate different states for real systems at different abstraction level.

Several new models of computation are discussed in [10] based on FSM to extend the modeling ability. In order to describe systems that require both control and computation, FSM and DFG are combined to form **Finite-State Machine with Datapath (FSMD)** . FSMD is the extended definition of a FSM by including a datapath with the set of datapath variables, input and outputs added. In order to get a concise behavior specification, a superstate is introduced to extend FSMD by merging programming language models to form the *superstate Finite-State Machine with Datapath (SFSMD)* in which each state can represent any number of clock cycles and the operations in each superstate are specified by a function or procedure written in a certain programming language, or by mathematical expressions. The SFSMD model is suitable for describing the behaviors for behavioral synthesis [10]. In order to eliminate the potential for the state and arc explosion, hierarchical and concurrent systems with FSM models, an extension of FSM model called *hierarchical concurrent finite-state machine (HCFSM),* is proposed by adding the support for hierarchy and concurrency. In the HCFSM each state can be further decomposed into a set of substates so that hierarchy can be modeled, and each state can also be decomposed into concurrent substates, which can execute concurrently and communicate through global variables. HCFSM model is well-suited to represent complex control systems without complex data structures [10].

2.1.4 Program State Machine

Program-state machine (PSM) [25] is an extension of FSM that supports both hierarchy and concurrency, and allows states to contain regular program code. It combines the description of both hardware and software models. The PSM model contains hierarchical program-states and a set of transition arcs. Program-states are either *composite* ones, which can be further decomposed into concurrent or sequential substates, or *leaf* ones, which are at the bottom of the hierarchy and contain computations described with programming language statements. Transition arcs are contained in a sequentially decomposed program-state to represent the sequencing between the program-substates. At the bottom of the hierarchy, there are those leaf states. A PSM can also overcome the primary limitation of programming languages, since it can model states explicitly [10]. PSM can be captured by both SpecCharts (an extension of VHDL) and SpecC (an extension of C) programming languages.

2.1.5 Transaction Level Modeling (TLM)

Transaction-level modeling (TLM) [12] is a well-accepted approach to model digital systems where the implementation details of the communication and functional units are abstracted and separated. At the transaction level, the emphasis is more on the functionality of the data transfers and less on their actual implementation. TLM abstracts away the low level system details about pins, wires and waveforms, which results in a model that executes dramatically faster than synthesizable models. However, this benefit generally comes at the price of low accuracy. In general, TLMs pose a trade-off between simulation speed and simulation accuracy. High simulation speed is traded in for low accuracy, and a high degree of accuracy comes at the price of low speed. Having both high speed and high accuracy at the same time requires sophisticated modeling and is only possible in special situations [24]. TLM is mainly used for communication simulation but lack the path to vertical integration of the models for implementation and synthesis. It is popular but not yet a well-defined MoC since it does not have any specific definition of the abstraction level nor transaction semantics. Moreover, TLM does not specify a well-defined MoC, but relies on the system design flow and the used SLDL to define the details of supported syntax and semantics.

2.2 Modern C-based SLDLs

System-level description languages (SLDL), like SpecC [10] and SystemC [12], are available for modeling and describing an embedded system at different levels of abstraction.

SpecC is a system level description language (as well as a system level design methodology). SpecC is a superset of ANSI-C. SpecC includes support for three computation models: concurrent sequential processes (CSP) for software, finite state machine with datapath (FSMD) for hardware and discrete event (DE) for protocols [10], by introducing new data structures, like behaviors, interfaces, and communication channels.

SystemC is the de-facto standard SLDL, supported by the Open SystemC Initiative. It is a C++ language library for system modeling. It builds systems from Verilog- and VHDL-like modules, each of which have a collection of I/O ports and instances of other modules or processes written in C++. It uses a discrete-event simulation model, and is quite flexible to interact with by general programming languages [4].

However, both C-based SLDLs do not define any details of an actual design flow. Moreover, there is a lack of efficient system-level formal models behind these SLDLs.

3 Problem Definition

For system-level design, the importance of abstract modeling cannot be over-emphasized. Proper abstraction and specification of the system model is a key to accurate and efficient estimation and the final successful implementation.

Digital circuit design provides a good example to show the importance of a well-defined model of computation. Designers describe the hardware components in hardware description languages (HDL) like VHDL and Verilog. Both languages have strong abilities to support different types of hardware structures and functionalities. By using the HDL, designers can use FSMs to model

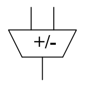
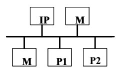
Abstraction Level	Schematics	Language	MoC	Tool
RTL		VHDL, Verilog	FSM, FSMD	Synopsys Design Compiler Cadence RTL Compiler ...
System		SpecC, SystemC	PSM, TLM (?) *ConcurrenC !*	SoC Environment [2] Synopsys System Studio ...

Table 1: System-level design in comparison with the well-established RTL design

synthesizable controllers or other parts of their design. FSM plays a crucial role in digital circuit design as a formal model behind the hardware description language. The FSM MoC allows the designer to describe their components in an efficient and accurate way. It also helps the synthesis tools to have a better understanding of the behavior and characteristics of the implemented circuit.

The importance of the model for system design is the same as for digital logic design. Due to the complexity of the system, which integrates multiple hardware components and various software processes together, a formal model is needed to clearly describe the critical features and ignore unnecessary details.

Note that commercial computer aided design (CAD) tools cannot synthesize all the VHDL / Verilog statements into a real hardware netlist. Special design guidelines are provided together with the CAD tools to restrict the designer from using specific syntax elements, or to prevent generation of improper logics, e.g. latches. For system-level design, guidelines are also needed for building systems by use of those SLDLs for efficient execution of the following design tasks by the tools.

Table 1 compares the situation in system-level design against the mature design methodology at the register-transfer level (RTL). RTL design is supported by the strong MoCs of FSM and FSMD, and well-accepted ”coding guidelines” exist for the HDLs VHDL and Verilog, so that established commercial tool chains can implement the described hardware. It is important to notice that here the MoC was defined first, and the coding style in the respective HDLs followed the needs of the MoC.

At the system level, on the other hand, we have the popular C-based SLDLs SystemC and SpecC which are more or less supported by early academic and commercial tools. As at RTL, the languages are restricted to a (small) subset of supported features, but these ”modeling guidelines” are not very clear. Moreover, the MoC behind these SLDLs is unclear. SpecC is defined in context of the PSM MoC [10], but so is SpecCharts [9] whose syntax is entirely different. For SystemC, one could claim TLM as its MoC [12], but a wide variety of interpretations of TLM exists.

We conclude that in contrast to the popularity of the C-based SLDLs for system-level modeling and validation, and the presence of early design flows supported by existing tools, the use of a well-defined and consistent system-level MoC is neglected or even completely ignored. As a result, serious restrictions are imposed on the usable (i.e. synthesizable and verifiable) subset of the supported SLDL. Without a clear MoC behind these syntactical ”modeling” guidelines, such restrictions appear almost arbitrary. Clearly, a well-defined and formal model is needed for the modern system description languages to improve and simplify the system design challenge.

4 ConcurrenC MoC

In the following, we discuss the close relationship and tight dependencies between SLDLs (i.e. syntax), their expressive abilities (i.e. semantics), and the abstract models they can represent. At the same time, we will point out that, in contrast to the large set of models the SLDL can describe, the available tools usually support only a small subset of these models. In order to avoid this discrepancy that clearly hinders the effectiveness of the system design methodology, we propose a novel MoC, called ConcurrenC, that aptly fits both the system modeling requirements and the capabilities of the supporting tool chain.

Generally speaking, we propose ConcurrenC as a system-level FSM extension with support for concurrency and hierarchy. As such, it falls into the program-state machine MoC category. The ConcurrenC model has clear separation of concerns on computation and communication.

In the realm of computation abstraction, the ConcurrenC model consists of blocks, channels, and interfaces, and fully supports structural and behavioral hierarchy. Blocks can be flexibly composed in space and time to execute sequentially, in parallel/pipelined fashion, or by use of state transitions. The blocks themselves are internally based on C, the most popular programming language for embedded system design.

In the realm of communication abstraction, ConcurrenC is intentionally restricted to a set of predefined channels that follow a typed message passing paradigm, rather than using user-defined freely programmable channels.

ConcurrenC is also platform-agnostic. It will not contain any details about the platform and hardware components used. Void of any implementation details other than functionality will make it possible for a ConcurrenC model to freely map to any platform in later design steps.

4.1 Relationship to C-based SLDLs

More specifically, ConcurrenC fits into the SpecC and SystemC SLDLs. Aimed at a formal model, ConcurrenC abstracts the embedded system features and provides clear guidelines for the designer to efficiently use the SLDLs to build a system. On the other hand, the ConcurrenC model is captured and described by using the SLDLs. In other words, ConcurrenC is represented in C-based SLDLs.

Figure 3 shows the relationship between the C-based SLDLs, SystemC and SpecC, and the MoC, ConcurrenC. ConcurrenC is a true subset of the MoCs that can be described by SpecC and SystemC. This implies that ConcurrenC contains only those model features that can be described by both SpecC and SystemC. For example, exception handling, i.e. interrupt and abortion, is supported in SpecC by using the *try-trap* syntax, but SystemC does not have the capability to handle exceptions explicitly. On the other hand, SystemC supports the feature of waiting for a certain time *and* for some events at the same time, but SpecC does not have such ability. As shown in Figure 3, such features which can only be supported by one SLDL, will not be included in the ConcurrenC model.

Moreover, ConcurrenC excludes some features that both SpecC and SystemC can support (the shadow overlap part in Figure 3). We exclude these to make the ConcurrenC model more concise and convenient for modeling and design space exploration. For example, as mentioned before, ConcurrenC will restrict its communication channels to a predefined library rather than allowing the user to define the channels freely by themselves. This allows tools to recognize the channels and implement them in optimal fashion.

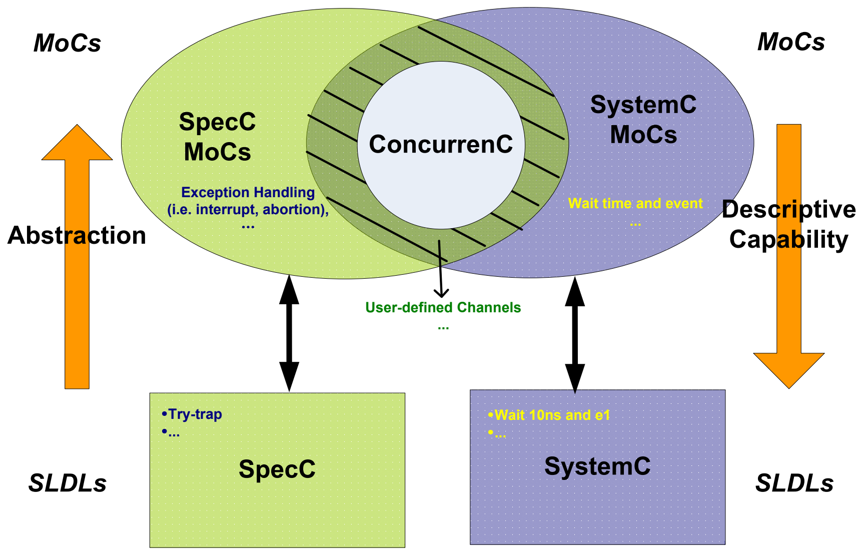


Figure 3: Relationship between C-based SLDLs: SystemC and SpecC, and MoC: ConcurrenC

4.2 ConcurrenC Features

A ConcurrenC Model can be visualized in four dimensions, as shown in Figure 4. There are three dimensions in space, and one in time. The spatial dimensions consist of two dimensions for structural composition of blocks and channels and their connectivity through ports and signals (X, Y coordinates in Figure 4), and one for hierarchical composition (Z-axis in Figure 4). The temporal dimension specifies the execution order of blocks in time, which can be sequential or FSM-like (thick arrows in Figure 4), parallel (dashed lines in Figure 4), or pipelined (dashed lines with arrows in Figure 4).

More detailed features of the proposed ConcurrenC MoC are discussed in the following subsections.

4.2.1 Communication & Computation Separation

Separating communication from computation allows "plug-n-play" features of the embedded system. In ConcurrenC, the communication contained in channels (shown as ellipses in Figure 4) is separated from the computation part contained in blocks (shown as rectangular blocks with rounded corners in Figure 4). The purpose of each statement in the model can be clearly identified whether

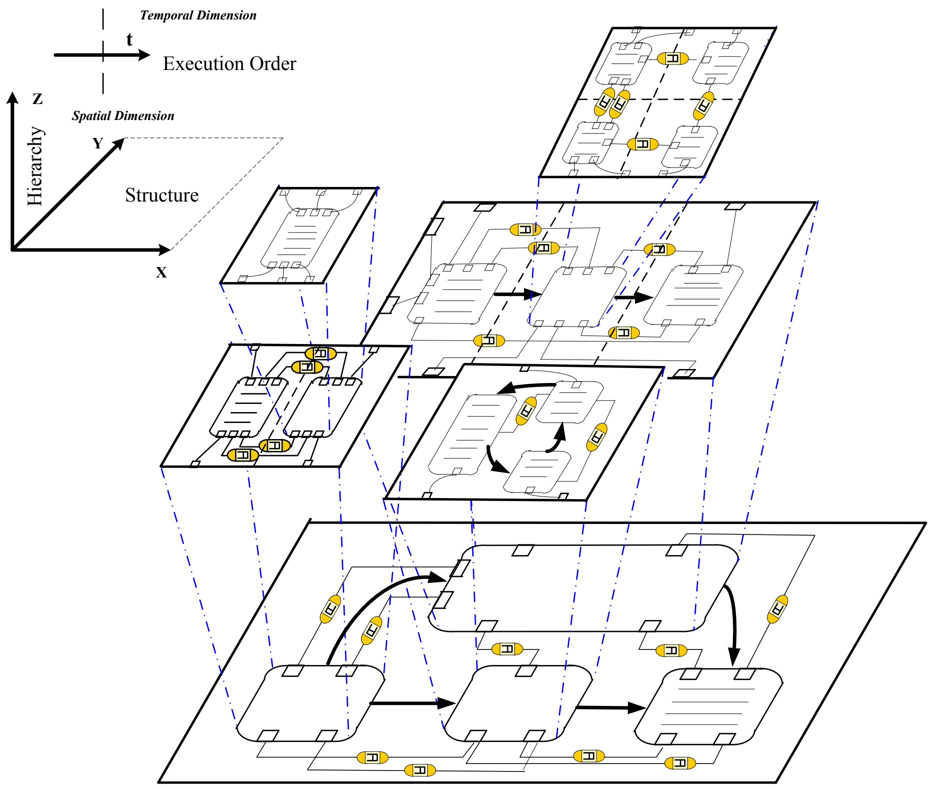


Figure 4: Visualization of a ConcurrenC Model in three spatial and one temporal dimensions

it is for communication or computation. This also helps for architecture refinement and hardware software partitioning.

4.2.2 Hierarchy

Hierarchy eliminates the potential explosion of the model size. It helps for comprehensible modeling of complex systems. This can be easily seen in Figure 4 where a single platform is much easier to understand than the entire system across all levels.

4.2.3 Concurrency

The need for concurrency is apparent. A common embedded system will have multiple hardware units work in parallel and cooperate through certain communication mechanisms. In Figure 4, parallel execution is shown by dashed border lines across a platform.

Pipelining is a special kind of parallelism. It is a common technique to improve the efficiency for both hardware and software. ConcurrenC explicitly supports the pipelining feature in order to provide simple and clear description of the intended system concurrency. In Figure 4, we show pipelining as dashed border lines combined with thick arrows indicating the pipeline flow.

4.2.4 Abstract Communications (Channels)

A predefined set of communication channels is available in ConcurrenC for the simplicity of system modeling and further synthesis work. We believe that the restriction to predefined channels not only avoids coding errors by the designer, but also simplifies the later refinement steps, since the channels can be easily recognized by the tools. More details will be discussed below in Section 4.3.

4.2.5 Timing

The execution time of the model should be evaluated to observe the efficiency of the system. System function calls written in C represent and maintain timing information since ConcurrenC models are internally pure C-based.

4.2.6 Execution

A model should be able to be executed in order to validate its correctness and obtain performance estimation. Since our ConcurrenC model can easily converted to SpecC and SystemC, the execution of the model is definitely possible.

4.3 Communication Channel Library

There are two kinds of communication channels: data transfer channel and synchronization channel. Channel properties, like blocking mode, data type support, data transfer support, flow direction, buffer size, and communication parties, should be considered to build a comprehensive but minimum set channel library. Moreover, in SpecC SLDL, we have three models of communication: shared memory, message passing (via function calls to channel methods), and protocol stack (via hierarchy channels). Since protocol stack is just a hierarchical channel, we could reduce the communication library to shared variables (for shared memory) and some kind of communication channels (for data transfer, message passing and protocol stack building), including channels for synchronization and data transfer.

As for data transfer channels, we limit the ConcurrenC channels to transfer data in FIFO fashion. FIFO channels are adopted in many MoCs, including KPN, SDF, and SHIM. In many cases, such channels make the model deterministic and allow static scheduling. For KPN, the buffer size is infinite (Q_∞), however, this is not practical for real-world applications despite of its powerful deterministic and deadlock-free property. For SDF, the buffer size is fixed (Q_n) which is more practical for real-world systems. Double handshake communication mechanism, which behaves in a rendezvous fashion, should also be supported. This can be seen as a FIFO with buffer size of zero (Q_0). Signals could be used to design a 1-N (broadcasting) channel. Furthermore, shared variables are regarded as a special kind of communication channel without any built-in synchronization that is often convenient (especially in software).

As for pure synchronization channels, mutex, semaphore, critical section, token, and barrier are often used. Mutex is a special semaphore with binary variable. The key to a critical section is to use a semaphore. Therefore, we can reduce both mutex and critical section to semaphores. Moreover, FIFO channel can be used to implement a semaphore.

We conclude that ConcurrenC should [1] support the following predefined communication channel library. There are five types of channels: Q_0, Q_n, Q_∞, signal, and shared variable. Here, Q stand for queue which behaves in FIFO fashion, and the indices $_0$, $_n$, $_\infty$ stands for zero, a certain finite number, and an infinite buffer size in the queue, respectively. Table 2 shows the proposed parameterized channel library.

Channel Type	Receiver	Sender	Buffer Size
Q_0	Blocking	Blocking	0
Q_n	Blocking	Blocking	n
Q_∞	Blocking	–	∞
Signal	Blocking	–	1
Shared Variable	–	–	1

Table 2: Parameterized Communication Channels

4.4 Relationship to KPN and SDF

With the features discussed above, it is quite straightforward to convert both KPN and SDF MoCs into ConcurrenC .

The conversion rules from KPN to ConcurrenC are listed as the pseudo-code in Algorithm 1.

```
Input: A general KPN model
Output: A ConcurrenC model with the same function
1: for all processes i ∈ KPN do
2:     make ConcurrenC blocks
3: end for
4: for all channels i ∈ KPN do
5:     make ConcurrenC channels of type $Q_\infty$
6: end for
7: keep the same connectivity in ConcurrenC as in KPN
8: If desired, group processes in hierarchy and size the channels for real-world implementation
```

Algorithm 1: Algorithm to convert KPN model into ConcurrenC

The conversion rules from SDF to ConcurrenC are listed as the pseudo-code in Algorithm 2.

[1] At this time, this decision is subject to further refinement in the future.

```
Input: A general SDF model
Output: A ConcurrenC model with the same function
1: for all actors $\in$ SDF do
2:     make ConcurrenC blocks
3: end for
4: for all arcs $\in$ SDF do
5:     make ConcurrenC channels of type $Q_n$ ($n$ is the size of the buffer)
6: end for
7: keep the same connectivity in ConcurrenC as in SDF.
8: If desired, group actor in hierarchy.
```

Algorithm 2: Algorithm to convert SDF model into ConcurrenC

As such, ConcurrenC is essentially a superset MoC of KPN and SDF. It is a versatile and convenient vehicle to implement KPN and SDF into SpecC or SystemC, i.e. by using ConcurrenC as the intermediate MoC since ConcurrenC is straightforward to implement via the well-established SLDLs. Moreover, the strong formal properties of KPN and SDF, such as deadlock-free guarantees (KPN) and static schedulability (SDF), are still inherited when modeled in ConcurrenC.

4.5 ConcurrenC-based Design Flow

The envisioned ConcurrenC is a MoC at system abstraction level. System features can be captured and execution is possible for validation and simulation by modeling a system in ConcurrenC and converting the model to SpecC and/or SystemC SLDL in order to perform fast and accurate design space exploration. Furthermore, based on this flexible model, a system can be mapped to a suitable target platform and implemented. Finally, the system can be synthesized to be a prototype implementation on an FPGA platform or a real MPSoC chip.

Figure 5 shows our envisioned design flow of ConcurrenC based system-level design.

5 Experiment

The Advanced Video Coding (AVC) standard H.264, also known as MPEG-4, is a real-world application for advanced video compression [19], [22]. Its high complexity, free availability, and industry-size make it an ideal, realistic and challenging example for system-level design. The H.264 standard includes both encoding and decoding for video streams with different resolution. We focus on the decoding part in this section. Figure 6 shows the basic block diagram of the H.264 decoder algorithm.

The input of the decoder is an H.264 stream file, while the output is an uncompressed YUV file. The compressed input video stream, from the network or other media and wrapped as the network abstraction layer (NAL) unit, is first entropy decoded and reordered. Then the decoded data is sent to inverse quantization and transformation, which means the output data are in time domain then. A picture frame is then rendered based on frame predictions. There are two types of frame prediction: intra-frame prediction (with motion compensation) and inter-frame prediction. How to predict depends on the frame types in the input video streams. At any time, a certain number of

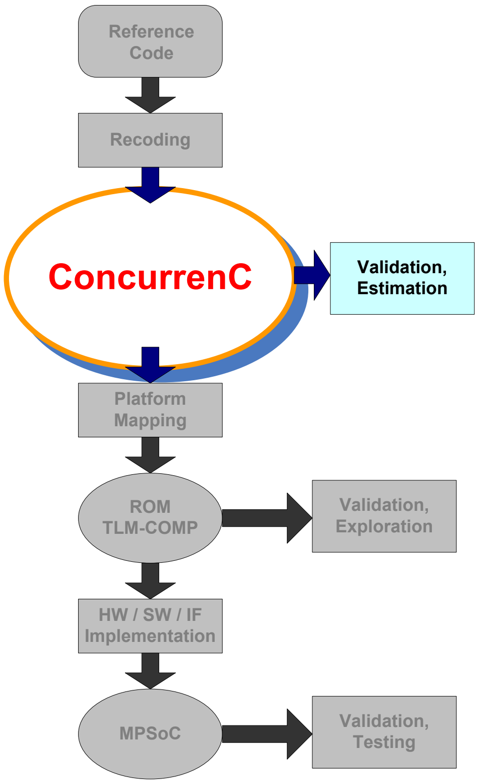


Figure 5: Envisioned ConcurrenC System Design Flow

decoded frames will be stored for inter-frame prediction since these are required by their following frames as references. In the final stage, the reconstructed video appears after the deblocking filter.

ConcurrenC modeling features can be easily applied to the H.264 decoder system. Figure 7 shows our ConcurrenC model of the H.264 decoder.

- **Hierarchy**: At the top level of the ConcurrenC model, there are three behavioral blocks: **stimulus, decoder, and monitor**. The **stimulus** reads the input yuv file, while the monitor receives and displays the decoded information, including signal-to-noise ratio (SNR), frame information, and system time, and writes those reconstructed frames into the output file.

  Decoder contains multiple blocks for concurrent slice decoding. A stream processing block prepares the settings, *n* decode units decode *n* slices in parallel, and the decoding synchronizer combines the decoded slices for output by the monitor. The number of the slice decoders is scalable depending on the number of slices contained in one frame of the input stream file. Inside the slice decode blocks, sub functional blocks are modeled for the detailed decoding tasks, e.g. entropy decoding & reordering, inverse quantization & transformation, motion compensation & Inter-frame prediction, deblock filtering, and buffer controlling. Blocks are in different colors for different levels in Figure 7. Hierarchical modeling allows convenient

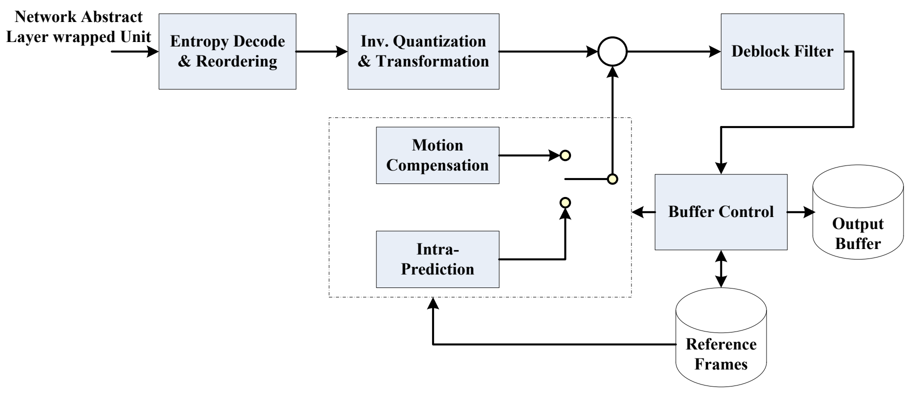

Network Abstract Layer wrapped Unit
Entropy Decode & Reordering
Inv. Quantization & Transformation
Deblock Filter
Motion Compensation
Intra-Prediction
Buffer Control
Output Buffer
Reference Frames

Figure 6: H.264 Decoder Algorithm Block Diagram

filename	boat.264				coastguard.264			
# macroblocks/frame	396				396			
# frames	73 (2.43 secs)				299 (9.97 secs)			
# slices/frame	4		8		4		8	
max # macroblocks/slice	150		60		150		60	
model type	seq	par	seq	par	seq	par	seq	par
host sim time (s)	4.223	4.258	4.557	4.550	12.191	12.197	12.860	12.846
estimated exec time (s)	11.13	4.43	11.49	**1.80**	18.78	**7.20**	20.31	**3.33**
speedup	1	2.51	1	6.38	1	2.61	1	6.10

Table 3: Simulation Results, H.264 Decoder modeled in ConcurrenC

and clear system description.

- **Concurrency**: [26] confirms that it is possible to have multiple slices in one frame being decoded at the same time. Parallelism is also usable for rendering (inter-prediction and intra-prediction) and filtering (deblock filter) stages inside the slice decoding [8].

  Consequently, our H.264 decoder model consists of multiple blocks for concurrent slice decoding for each picture frame. Additional parallelism is exploited inside the rendering and filtering stages, considering the properties of the filtering and transformation algorithms.

- **Communication Abstraction**: FIFO channels and shared variables are used for communication in our H.264 decoder model. FIFO queues are used for data exchange between different blocks. For example, the decoder synchronizer sends the decoded frame via a FIFO channel to the monitor for output. Shared variables, i.e. reference frames, are used to simplify the coordination for decoding multiple slices in parallel.

- **Timing** The decoding time can be observed by using timing function calls written in C (since

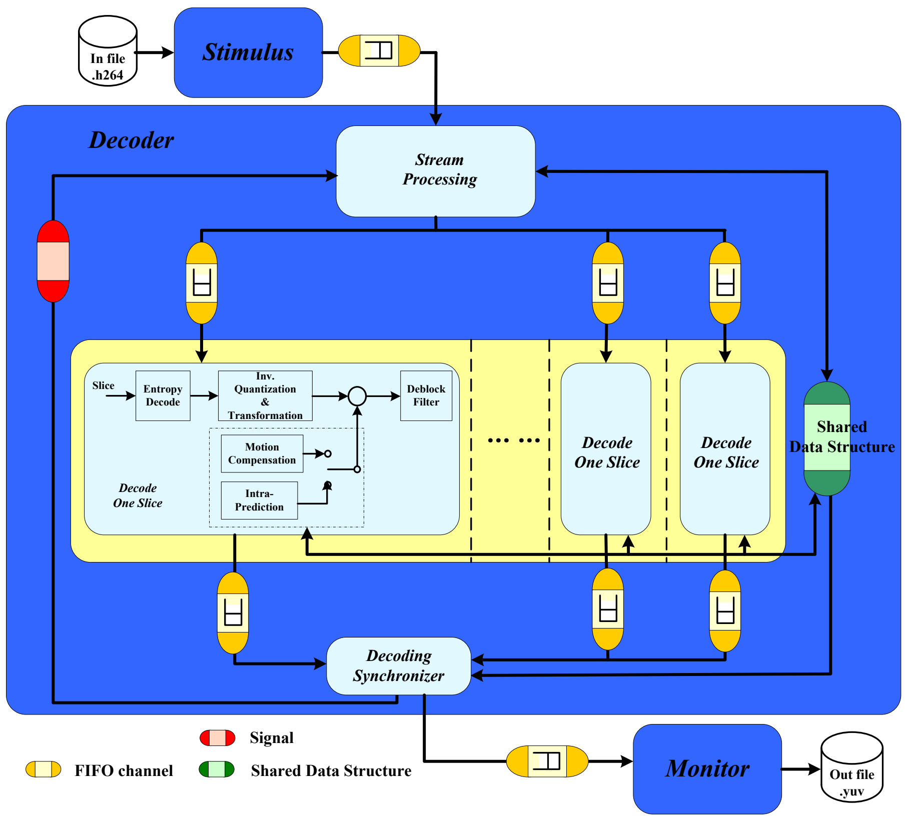


Figure 7: ConcurrenC H.264 Decoder Diagram

the ConcurrenC model is internally C-based). We have obtained the estimated execution time for different hardware architectures by using simulation and profiling tools of the SpecC SLDL.

- **Execution** We have successfully converted and executed our model using the SoC Environment [2].

Table 3 shows the simulation results of our H.264 decoder modeling in ConcurrenC. The model has been converted to SpecC and is simulated on a host machine with Intel(R) Pentium(R) 4 CPU at 3.00GHz. We have tested the decoder with two stream files, one with 73 frames, and the other with 299 frames. For each test stream, we have created two types of streams, 4 slices and 8 slices per frame. We run the model by decoding the input streams in two ways: slice by slice (seq model), and slices in one frame concurrently (par model). The estimated execution time is measured by annotating the timing information manually into the model according to the profiling and estimation results generated by the SCE tool with ARM7TDMI 400 MHz processor mapping. The simulation results show that the application modeled by ConcurrenC is scalable. High speedup is gained by decoding the slices in the parallel fashion. We can expect that it is possible to decode the test streams in three platform configurations in real-time when using the system running on a ARM7TDMI CPU with 400 MHz (bold times).

6 Conclusion

In this report, we propose a new model of computation, ConcurrenC, that is suited to C-based SLDLs. ConcurrenC is a concurrent, hierarchical system model of computation with abstractions of both communication and computation. The features of the new MoC and its relationship with C-based SLDLs have been discussed in detail. A real-world driver application, H.264 decoder, which is a suitable application with great complexity and industrial size, is used to demonstrate how the proposed ConcurrenC features match the system design requirements.

We conclude that the proposed ConcurrenC will be a new MoC that aptly fits the system-level abstraction and can be captured by modern C-based SLDLs. As such, we expect ConcurrenC to be a practical solution to improving embedded system modeling and design.

Conceptually, ConcurrenC fills the gap between the theoretical MoCs KPN and SDF, and the practical SLDLs SpecC and SystemC.

6.1 Future Work

This report is only the beginning of a challenging research project. While the basic ideas and foundations have been laid, there are several milestones ahead. Future work includes the definition of a formal execution semantics of ConcurrenC, the design and implementation of a robust data structure and software framework, and the integration with existing or future design flows and tool suites. We look forward to solving these exciting challenges.

Acknowledgment

This work has been supported in part by funding from the National Science Foundation under research grant NSF Award #0747523. The authors thank the NSF for the valuable support.

References

[1] B.Bhattacharya and S.S.Bhattacharyya. Parameterized Dataflow Modeling of DSP Systems. In *International Conference on Acoustics, Speech, and Signal Processing*, Istanbul, Turkey, June 2000.

[2] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski. System-on-chip environment: a specc-based framework for heterogeneous mpsoc design. *EURASIP J. Embedded Syst.*, 2008(3):1–13, 2008.

[3] E.A.Lee and D. Messerschmitt. Synchronous Data Flow. 75(9):1235–1245, September 1987.

[4] S. A. Edwards. Design Languages for Embedded Systems. Technical report, Columbia University, New York, 2003.

[5] S. A. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. *Proc. of the IEEE*, 85(3), Mar. 1997.

[6] S. A. Edwards and O. Tardieu. SHIM: A Deterministic Model for Heterogeneous Embedded Systems. *IEEE Transactions on VLSI Systems*, 14(8):854–867, 2006.

[7] M. Engels, G. Bilsen, R. Lauwereins, and J. Peperstraete. Cyclo-static data flow: Model and implementation. *Proc. 28th Asilomar Conf. on Signals, Systems, and Computers*, pages 503–507, 1994.

[8] K. Fleming, C.-C. Lin, N. Dave, Arvind, G. Raghavan, and J. Hicks. H.264 decoder: A case study in multiple design points. In *Formal Methods and Models for Co-Design, 2008. MEMOCODE 2008. 6th ACM/IEEE International Conference on*, June 2008.

[9] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice Hall, 1994.

[10] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.

[11] A. Girault, B. Lee, and E. A. Lee. Hierarchical Finite State Machines with Multiple Concurrency Models. *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems (TCAD)*, 18(6), June 1999.

[12] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[13] International Semiconductor Industry Association. International Technology Roadmap for Semiconductors (ITRS). http://www.itrs.net, 2007.

[14] J.T.Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model*. PhD thesis, Department of EECS, University of California, Berkeley, CA 94720, 1993.

[15] G. Kahn. The semantics of a simple language for parallel programming. *Information Processing*, pages 471–475, 1974.

[16] E. A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems (TCAD)*, 17(12), Dec. 1998.

[17] MoC wikipedia. http://en.wikipedia.org/wiki/Model_of_computation.

[18] T. Murata. Petri nets: Properties, analysis and applications. 77(4):541–580, April 1989.

[19] J. V. T. of ITU-T and I. J. 1. *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC)*. Document JVT-G050r1, 2003.

[20] T. M. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, Electrical Engineering and Computer Science, University of California, Berkeley, December 1995.

[21] J. L. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3):223–252, September 1977.

[22] I. E. G. Richardson. H.264/MPEG-4 Part 10 White Paper. http://www.vcodex.com/, 2002.

[23] A. Sangiovanni-Vincentelli. Quo Vadis SLD: Reasoning about Trends and Challenges of System-Level Design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[24] G. Schirner and R. Dömer. Result Oriented Modeling - A Novel Technique for Fast and Accurate TLM. *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems (TCAD)*, 26(9):1688–1699, 2007.

[25] F. Vahid, S. Narayan, and D. D. Gajski. SpecCharts: A VHDL frontend for embedded systems. *IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems (TCAD)*, 14(6):694–706, June 1995.

[26] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[27] G. Zhou. Dynamic Dataflow Modeling in Ptolemy II. *Technical Memorandum No. UCB/ERL M05/2, University of California, Berkeley, CA, 94720, USA*, December 2004.