

# ConcurrenC: A New Approach towards Effective Abstraction of C-based SLDLs

Weiwei Chen and Rainer Doemer

Center for Embedded Computer Systems  
University of California, Irvine  
weiwei.chen@uci.edu, doemer@uci.edu

**Abstract.** Embedded system design in general can only be successful if it is based on a suitable 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, such as SystemC and SpecC, are popular in system-level modeling and validation, current tool flows impose serious restrictions on the synthesizable subset of the supported SLDL. A properly aligned and clean system-level MoC is often neglected or even ignored. In this paper, we motivate the need for a well-defined MoC in embedded 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 outline a new approach, called *ConcurrenC*, that defines a true system level of abstraction, aptly fits system modeling requirements, and can be expressed precisely in both SystemC and SpecC. Using the case study of a H.264 video decoder, we demonstrate how the ConcurrenC approach meets the needs and characteristics of a industry size embedded application.

## 1 Introduction

With applications ranging from portable media players to real-time automotive applications, the complexity of embedded systems is growing rapidly with the increasing number of integrated components that have to cooperate properly and concurrently. Embedded systems also have tight constraints on size, power, and price.

According to the 2007 edition of the International Technology Roadmap for Semiconductors (ITRS) [7], system-level design is a promising solution to improve the design productivity by moving up to a higher level of abstraction. A new modeling approach is needed to enable system design, including simulation, estimation, synthesis, verification, implementation, and design space exploration.

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

After a brief overview about existing MoCs in related work, we discuss the need for a new approach on C-based system design in Section 3. We then outline our approach, called *ConcurrenC*, in Section 4. To demonstrate ConcurrenC, we study in Section 5 a

H.264 video decoding application and show how the inherent features and characteristics of this application can be naturally reflected in the proposed Concurrent MoC.

The contribution of this paper is the identification of needs and requirements of a new MoC that enables effective abstraction of designs specified in C-based SLDLs.

## 2 Related Work

There has been considerable effort on many MoCs and SLDLs, as well as existing system design flows.

A **Model of Computation (MoC)** is a formal definition of the set of allowable operations used in computation and their respective costs. This defines the behavior of a system at a certain abstraction level to reflect the essential system features. Many different MoCs have been proposed for different domains. Overviews can be found in [4] and [10].

**Kahn Process Network (KPN)** is a deterministic MoC where processes are connected by unbounded FIFO channels to form a network [9]. **Dataflow Process Network (DFPN)** [11] is a special case of KPN in which the communication buffers are bounded. **Synchronous dataflow (SDF)** is an extended MoC from DFPN that allows static scheduling. While these MoCs are popular for modeling signal processing applications, they are not well-suited for controller applications.

**Dataflow Graph (DFG)** and its derivatives are MoCs for describing computational intensive systems [1]. Combined with **Finite State Machine (FSM)**, which is popular for describing control systems, FSM and DFG form **Finite-State Machine with Data-path (FSMD)** in order to describe systems requiring both control and computation [5]. **Program-state machine (PSM)** [4] is a FSM extension that supports both hierarchy and concurrency and allows states to contain regular program code.

**Transaction-level modeling (TLM)** [6] is a well-accepted approach to model digital systems where the details of communication are abstracted. Unfortunately, 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.

Modern C-based SLDLs, like SystemC [6] and SpecC [5], are available for modeling and describing an embedded system at different levels of abstraction. Both include support for describing several MoCs, including PSM, FSMD, TLM, and general discrete event (DE) simulation. However, neither language defines a formal model behind the plain syntax and execution semantics.

## 3 Problem Definition



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

Register-Transfer Level (RTL) design is a good example to show the importance of a well-defined model of computation. Designers describe hardware components in hardware description languages (HDL), i.e. VHDL and Verilog. Both languages have

strong abilities to support different types of hardware structures and functionalities. By using the HDL, designers use FSMs to model controllers or other parts of their design. Thus, FSM plays a crucial role as a formal model behind the languages. In other words, the FSM model in the mind of the designer is described syntactically in the VHDL or Verilog language.

Note that commercial computer aided design (CAD) tools cannot synthesize all the VHDL / Verilog statements. Instead, special design guidelines are provided to restrict the use of specific syntax elements, or to prevent generation of improper logics, e.g. latches.

The importance of the model in system design is the same as in RTL. Table 1 compares the situation at the system level against the mature design methodology at the RTL. RTL design is supported by the strong MoCs of FSM and FSMD, and well-accepted *coding guidelines* exist for 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 HDL followed the needs of the MoC.

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

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

At the Electronic System Level (ESL), 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 [2, 3]. 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 [5], but so is SpecCharts [4] whose syntax is entirely different. For SystemC, one could claim TLM as its MoC [6], but a wide variety of interpretations of TLM exists.

We can conclude that in contrast to the popularity of the C-based SLDLs for ESL modeling and validation, and the presence of existing design flows implemented by early tools, the use of a well-defined and clear system-level MoC is neglected. Instead, serious restrictions are imposed on the usable (i.e. synthesizable and verifiable) subset of the supported SLDL. Without a clear MoC behind these syntactical guidelines, computer-aided system design is difficult. Clearly, a well-defined and formal MoC is needed to attack and solve the ESL design challenge.

## 4 ConcurrentC MoC

We now 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. We will point out that, in contrast to the large set of models the SLDL can describe,

the available tools support only a subset of these models. To avoid this discrepancy that clearly hinders the effectiveness of any ESL methodology, we propose a novel MoC, called ConcurrnC, that aptly fits the system modeling requirements and the capabilities of the supporting tool chain and languages.

Generally speaking, ConcurrnC should be a system-level FSM extension with support for concurrency and hierarchy. As such, it falls into the PSM MoC category. The ConcurrnC model needs clear separation of concerns on computation and communication. In the realm of structure abstraction, a ConcurrnC 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. Blocks themselves are internally based on C, the most popular programming language for embedded applications. In the realm of communication abstraction, we intentionally use a set of predefined channels that follow a typed message passing paradigm rather than using user-defined freely programmable channels.

### Relationship to C-based SLDLs

More specifically, ConcurrnC is tailored to the SpecC and SystemC SLDLs. ConcurrnC abstracts the embedded system features and provides clear guidelines for the designer to efficiently use the SLDLs to build a system. In other words, the ConcurrnC model is captured and described by using the SLDLs.

Fig. 1 shows the relationship between the C-based SLDLs, SystemC and SpecC, and the MoC, ConcurrnC. ConcurrnC is a true subset of the models that can be described by SpecC and SystemC. This implies that ConcurrnC contains only the model features which can be described by both languages. 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 such exceptions. On the other hand, SystemC supports the feature for waiting a certain time *and* for some events at the same time, which SpecC does not support. As shown in Fig. 1, features that are only supported by one SLDL will not be included in the ConcurrnC model.

Moreover, ConcurrnC excludes some features that both SpecC and SystemC support (the shadow overlap area in Fig. 1). We exclude these to make the ConcurrnC model more concise for modeling. For example, ConcurrnC will restrict its communication channels to a predefined library rather than allowing the user to define arbitrary channels by themselves. This allows tools to recognize the channels and implement them in optimal fashion.

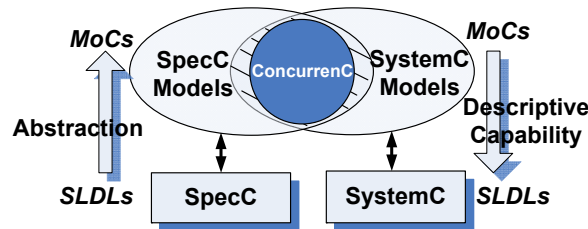
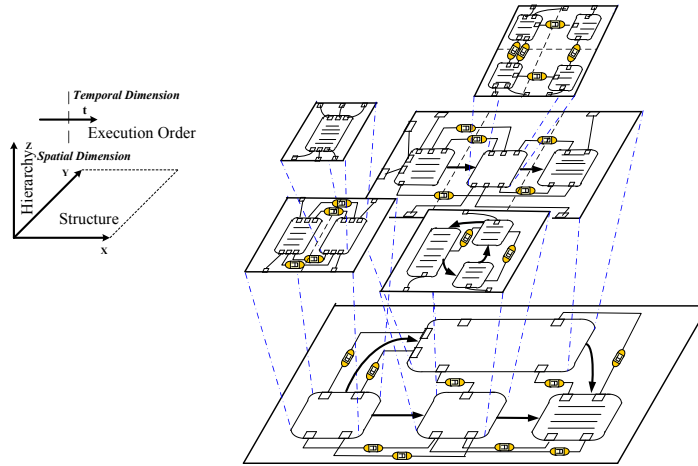


Fig. 1. Relationship between C-based SLDLs SystemC and SpecC, and MoC ConcurrnC

## ConcurrenC Features

A ConcurrenC Model can be visualized in four dimensions as shown in Fig. 2. 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), and one for hierarchical composition (Z-axis). The temporal dimension specifies the execution order of blocks in time, which can be sequential or FSM-like (thick arrows), parallel (dashed lines), or pipelined (dashed lines with arrows) in Fig. 2.



**Fig. 2.** Visualization of a ConcurrenC Model in three spatial and one temporal dimensions

The detailed features of the proposed ConcurrenC MoC are listed below:

- **Communication & Computation Separation** Separating communication from computation allows “plug-n-play” features of the embedded system [5]. In ConcurrenC, the communication contained in channels is separated from the computation part contained in blocks so that the purpose of each statement in the model can be clearly identified whether it is for communication or computation. This also helps for architecture refinement and hardware/software partitioning.
- **Hierarchy** Hierarchy eliminates the potential explosion of the model size and significantly simplifies comprehensible modeling of complex systems.
- **Concurrency** The need for concurrency is obvious. A common embedded system will have multiple hardware units work in parallel and cooperate through specified communication mechanisms. ConcurrenC also supports pipelining in order to provide a simple and explicit description of the pipelined data flow in the system.
- **Abstract Communications (Channels)** A predefined set of communication channels is available in ConcurrenC. 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.
- **Timing** The execution time of the model should be evaluable to observe the efficiency of the system. Thus, ConcurrenC supports wait-for-time statements in similar fashion as SystemC and SpecC.

- **Execution** The model must be executable in order to show its correctness and obtain performance estimation. Since a ConcurrnC model can be converted to SpecC and SystemC, the execution of the model is definitely possible.

### Communication Channel Library

For ConcurrnC, we envision two type of channels, channels for synchronization and data transfer. For data transfer, ConcurrnC limits the channel to transfer data in FIFO fashion (as in KPN and SDF). In many cases, these channels make the model deterministic and allow static scheduling. For KPN-like channels, the buffer size is infinite ( $Q_\infty$ ) which makes the model deadlock free but not practical. For SDF-like channels, the buffer size is fixed ( $Q_n$ ). Double-handshake mechanism, which behaves in a rendezvous fashion, is also available as a FIFO with buffer size of zero ( $Q_0$ ). Signals are needed to design a 1-N (broadcasting) channel. Furthermore, shared variables are allowed as a simple way of communication that is convenient especially in software. Moreover, FIFO channels can be used to implement semaphore which is the key to build synchronization channels. In summary, ConcurrnC supports the predefined channel library as shown in Table 2.

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

**Table 2.** Parameterized Communication Channels

## 5 Experiment

In order to demonstrate the feasibility and benefits of the ConcurrnC approach, we use the Advanced Video Coding (AVC) standard H.264 decoding algorithm [8] as driver application to evaluate the modeling features. Our H.264 decoder model is of industrial size, consisting of about 30 thousand lines of code. The input of the decoder is an H.264 stream file, while the output is a YUV file.

ConcurrnC features can be easily used to model the H.264 decoder, see Fig. 3.

- **Hierarchy:** At the top level of the ConcurrnC 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 stream including signal-to-noise ratio (SNR), system time, and writes the 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 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, functional sub-blocks are modeled for the detailed decoding tasks. Hierarchical modeling allows convenient and clear system description.

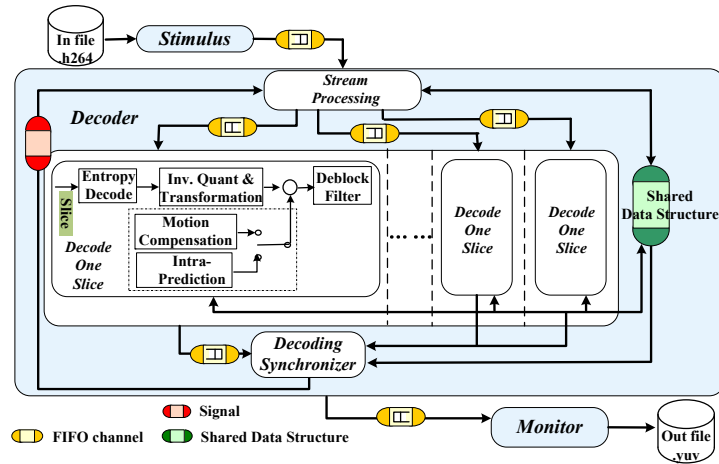


Fig. 3. Proposed H.264 Decoder Block Diagram

- **Concurrency:** [12] confirms that multiple slices in one frame are possible to be decoded concurrently. Consequently, our H.264 decoder model consists of multiple blocks for concurrent slice decoding in one picture frame<sup>1</sup>.
- **Communication:** 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 wait-for-time statements in the modeled blocks. We have obtained the estimated execution time for different hardware architectures by using simulation and profiling tools of the SLDLs.
- **Execution:** We have successfully converted and executed our model in SpecC using the SoC Environment [2].

Table 3 shows the simulation results of our H.264 decoder modeling in ConcurrnC. The model is simulated on a PC machine with Intel(R) Pentium(R) 4 CPU at 3.00GHz. Two stream files, one with 73 frames, and the other with 299 frames are tested. For each test file, we 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 annotated timing information according to the estimation results generated by SCE with a ARM7TDMI 400 MHz processor mapping. Our simulation results show that the parallelism of the application modeled in ConcurrnC is scalable. We can expect that it is possible to decode three of the test streams in real-time (bold times).

## 6 Conclusion

In this paper, we have discussed the relationship between C-based system description languages and the abstract design models they describe. We argue that a new model

<sup>1</sup> We should emphasize that this potential parallelism was not apparent in the original C code. It required serious modeling effort to parallelize the slice decoders for our model.

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	<b>1.80</b>	18.78	<b>7.20</b>	20.31	<b>3.33</b>
speedup	1	2.51	1	6.38	1	2.61	1	6.10

**Table 3.** Simulation Results, H.264 Decoder modeled in ConcurrnC

of computation is needed behind the syntax of the languages and have outlined a new model of computation, ConcurrnC. ConcurrnC is a concurrent, hierarchical system model of computation with abstraction of both communication and computation, that fits the requirements of both SpecC and SystemC SLDLs. A real-world driver application, H.264 decoder is used to demonstrate how the proposed ConcurrnC approach matches the system modeling requirements.

While we leave the detailed formal modeling for future work, the contribution of this paper is a practical approach at abstract system modeling that fills the gap between the theoretical MoCs KPN and SDF, and the practical SLDLs SpecC and SystemC.

## Acknowledgment

This work has been supported in part by funding from the National Science Foundation (NSF) under research grant NSF Award #0747523. The authors thank the NSF for the valuable support. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## References

1. T. DeMarco. Structured analysis and system specification. pages 409–424, 1979.
2. R. Doemer, 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. Embedded System Environment. <http://www.cecs.uci.edu/~ese/>.
4. D. D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice Hall, 1994.
5. D. D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Design Methodology*. Kluwer Academic Publishers, 2000.
6. T. Groetker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.
7. International Semiconductor Industry Association. International Technology Roadmap for Semiconductors (ITRS). <http://www.itrs.net>, 2007.
8. Joint Video Team of ITU-T and ISO/IEC JTC 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.
9. G. Kahn. The Semantics of a Simple Language for Parallel Programming. *Information Processing*, pages 471–475, 1974.
10. E. A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 17(12), Dec. 1998.



11. T. M. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, Electrical Engineering and Computer Science, University of California, Berkeley, December 1995.
12. 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.