Chapter 2
The ConcurrenC Model of Computation

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 chapter, we propose a new MoC, called *ConcurrenC*, that defines a system-level of abstraction, fits system modeling requirements, and can be expressed in both SystemC and SpecC SLDLs [CD09].

2.1 Motivation

For system-level design, the importance of abstract modeling cannot be overrated. Proper abstraction and specification of the system model is a key to an 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 MoC. Designers describe hardware components in hardware description languages (HDL), i.e., VHDL and Verilog. Both languages have strong capabilities to support different types of hardware structures and functionalities. By using the HDL, designers use Finite State Machines (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 to implement a hardware design.

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.

W. Chen, *Out-of-order Parallel Discrete Event Simulation for Electronic System-level Design*, DOI 10.1007/978-3-319-08753-5_2

Table 2.1 System-level design in comparison with the well-established RTL design

Abstraction level	Schematics	Language	MoC	Tool
RTL		VHDL, Verilog	FSM, FSMD	Synopsys design compiler, cadence RTL compiler...
ESL		SpecC, SystemC	PSM, TLM (?) *ConcurrenC !*	SoC environment [DGP+08], synopsys system studio...

The importance of the model in system design is the same as in RTL. Table 2.1 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 the VHDL and Verilog languags, 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.

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 [DGP+08, ESE]. 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 Program State Machine (PSM) MoC [GZD+00], but so is SpecCharts [GVNG94] whose syntax is entirely different. For SystemC, one could claim Transaction Level Model (TLM) as its MoC [GLMS02], 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 tackle the ESL design challenge.

2.2 Models of Computation

Edwards et al. argue in [ELLSV97] 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 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 [MoC]. This defines the behavior of the system is at 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 [GVNG94, LSV98].

Kahn Process Network (KPN) is a deterministic MoC where processes are connected by unbounded FIFO communication channels to form a network [Par95]. *Dataflow Process Network* (*DFPN*) [Par95], a special case of *KPN*, is a kind of MoC in which dataflow can be drawn in graphs as process network and the size of the communication buffer is bounded. *Synchronous dataflow* (*SDF*) [LM87], *Cyclo-static dataflow* (*CSDF*) [BELP96], *Heterochronous dataflow* (*HDF*) [GLL99], *Parameterized Synchronous dataflow* (*PSDF*) [BB00], *Boolean dataflow* (*BDF*) [Buc93], and *Dynamic dataflow* (*DDF*) [Zho04] are extended MoCs from the *DFPN* to provide the features like static scheduling, predetermined communication patterns, finite state machine (FSM) extension, reconfiguration, boolean modeling, and dynamic deadlock and boundedness analysis. These MoCs are popular for modeling signal processing applications but not suited for controller applications.

Software/hardware integration medium (SHIM) [ET06] is a concurrent asynchronous deterministic model, which is essentially an effective KPN with rendezvous communication for heterogeneous-embedded systems.

Petri Net [Pet77], an abstract, formal model of information flow, is a state-oriented hierarchical model, especially for the systems that being concurrent, distributed, asynchronous, parallel, non-deterministic or stochastic activities. However, it is uninterpreted and can also quickly become incomprehensible with any system complexity increase.

Dataflow Graph (DFG) and its derivatives are MoCs for describing computational intensive systems [GZD+00]. It is very popular for describing digital signal processing (DSP) components but is not suitable to represent control parts which are commonly found in most programming languages.

Combined with **Finite State Machine (FSM)** which is popular for describing control systems, *FSM* and *DFG* form *Finite State Machine with Datapath* (*FSMD*) in order to describe systems requiring both control and computation. *Superstate Finite-State Machine with Datapath* (*SFSMD*) and *hierarchical concurrent finite-state machine with Datapath* (*HCFSMD*) are proposed based on FSMD to support the hierarchical description ability with concurrent system features for behavioral synthesis.

Program State machine (PSM) [GVNG94] is an extension of *FSMD* that supports both hierarchy and concurrency, and allows states to contain regular program code.

Transaction-level modeling (TLM) [GLMS02] is a well-accepted approach to model digital systems where the implementation details of the communication and functional units are abstracted and separated. TLM abstracts away the low level system details so that executes dramatically faster than synthesizable models. However, high simulation speed is traded in for low accuracy, while a high degree of accuracy comes at the price of low speed. 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.

2.3 ConcurrenC MoC

Although we have the mature and industrially used SLDLs, like SpecC and SystemC, we do not have a formal model for both of them. In this section, we will 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. 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 *ConcurrenC*, that fits the system modeling requirements and the capabilities of the supporting tool chain and languages.

Generally speaking, *ConcurrenC* should be a system-level FSM extension with support for concurrency and hierarchy. As such, it falls into the PSM MoC category. The *ConcurrenC* model needs clear separation of concerns on computation and communication. In the realm of structure abstraction, a *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. Blocks themselves are internally based on C, which 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.

2.3.1 Relationship to C-based SLDLs

More specifically, *ConcurrenC* is tailored to the SpecC and SystemC SLDLs. *ConcurrenC* 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 *ConcurrenC* model can be captured and described by using the SLDLs.

Figure 2.1 shows the relationship between the C-based SLDLs, SystemC and SpecC, and the MoC, *ConcurrenC*. *ConcurrenC* is a true subset of the models that can be described by SpecC and SystemC. This implies that *ConcurrenC* 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

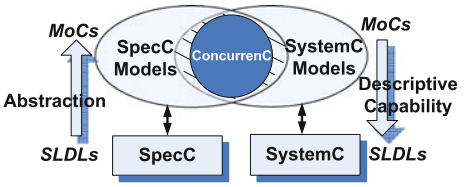

Fig. 2.1 Relationship between C-based SLDLs SystemC and SpecC, and MoC ConcurrenC

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. 2.1, features that are only supported by one SLDL will not be included in the *ConcurrenC* model.

Moreover, *ConcurrenC* excludes some features that both SpecC and SystemC support (the shadow overlap area in Fig. 2.1). We exclude these to make the *ConcurrenC* model more concise for modeling. For example, *ConcurrenC* 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 an optimal fashion.

2.3.2 ConcurrenC Features

A *ConcurrenC* Model can be visualized in four dimensions as shown in Fig. 2.2. 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

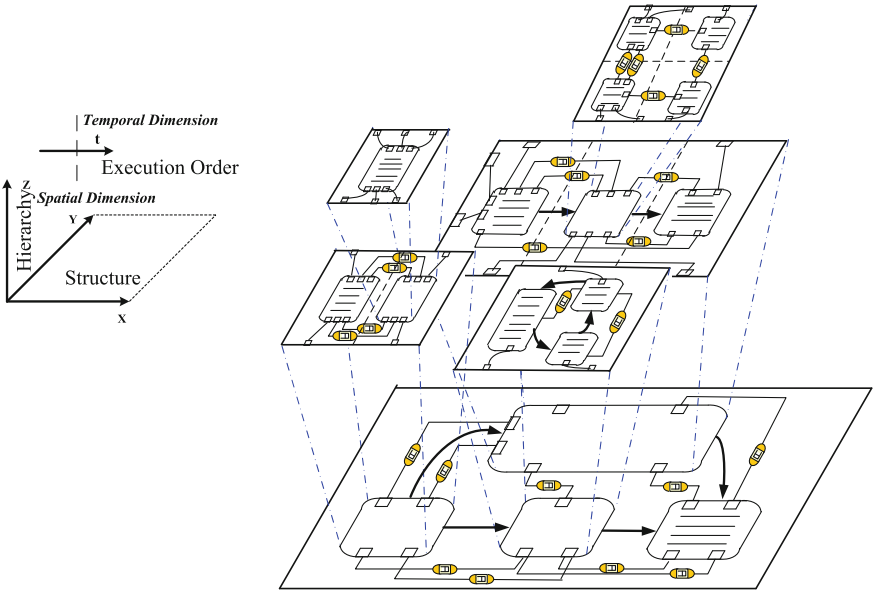

Fig. 2.2 Visualization of a ConcurrenC Model in three spatial and one temporal dimensions

blocks in time, which can be sequential or FSM-like (thick arrows), parallel (dashed lines), or pipelined (dashed lines with arrows) in Fig. 2.2.

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

- **Communication and Computation Separation**. Separating communication from computation allows "plug-n-play" features of the embedded system [GZD+00]. 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 *ConcurrenC* model can be converted to SpecC and SystemC, the execution of the model is thus possible.

2.3.3 Communication Channel Library

For *ConcurrenC*, we envision two type of channels, channels for synchronization and data transfer. For data transfer, *ConcurrenC* 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 ($\mathbf{Q}_\infty$) which makes the model deadlock free but not practical. For SDF-like channels, the buffer size is fixed ($\mathbf{Q}_n$). Double-handshake mechanism, which behaves in a rendezvous fashion, is also available as a FIFO with buffer size of zero ($\mathbf{Q}_0$). Signals are needed to design a $1-\mathrm{N}$ (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, *ConcurrenC* supports the predefined channel library as shown in Table 2.2.

Table 2.2 Parameterized communication channels

Channel type	Receiver	Sender	Buffer size
Q_0	Blocking	Blocking	0
Q_n	Blocking	Blocking	n
Q_∞	Blocking	–	∞
Signal	Blocking	–	1
Shared variable	–	–	1

2.3.4 Relationship to KPN and SDF

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

The conversion rules from KPN (SDF) to *ConcurrenC* are:

- ∀ processes ∈ KPN (SDF): convert into *ConcurrenC* blocks.
- ∀ channels ∈ KPN (SDF): convert into *ConcurrenC* channels of type Q_∞ (Q_n).
- Keep the same connectivity in *ConcurrenC* as in KPN (SDF).
- If desired, group blocks in hierarchy and size the KPN channels for real-world implementation.

The conversion rules from SDF to *ConcurrenC* are:

- ∀ actors ∈ SDF: convert into *ConcurrenC* blocks.
- ∀ arcs ∈ SDF: convert into *ConcurrenC* channels of type Q_n where n is the size of the buffer.
- keep the same connectivity in *ConcurrenC* as in SDF.
- If desired, group blocks in hierarchy.

As such, *ConcurrenC* is essentially a superset MoC of KPN and SDF. Also it becomes possible to implement KPN and SDF into SpecC and SystemC by using *ConcurrenC* as the intermediate MoC. Moreover, note that *ConcurrenC* inherits the strong formal properties of KPN and SDF, such as static schedulability and deadlock-free guarantees.

2.4 Case Study

In order to demonstrate the feasibility and benefits of the *ConcurrenC* approach, we use the Advanced Video Coding (AVC) standard H.264 decoding algorithm [Joi03] as a driver application to evaluate the modeling features. Our H.264 decoder model is of industrial size, consisting of about 40,000 lines of code. The input of the decoder is an H.264 stream file, while the output is a YUV file.

ConcurrenC features can be easily used to model the H.264 decoder, see Fig. 2.3.

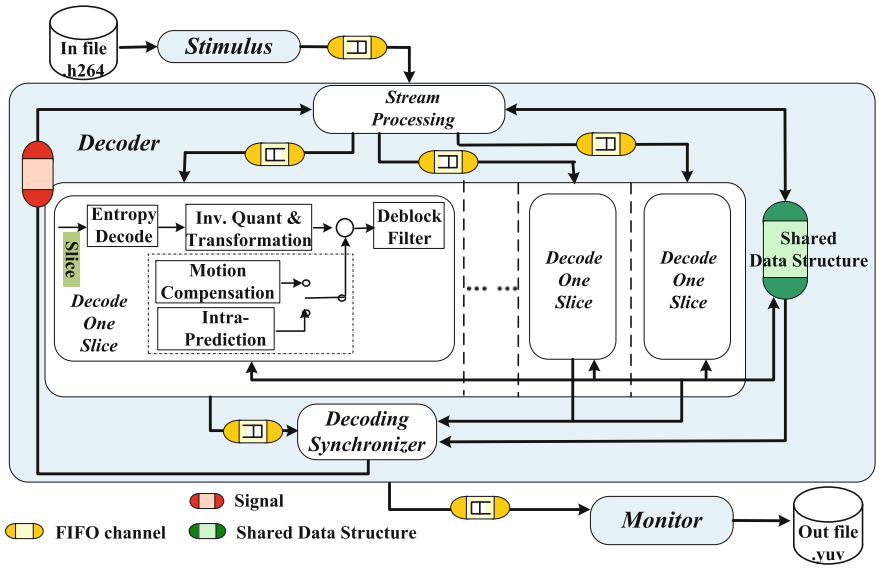

Fig. 2.3 Proposed H.264 decoder block diagram

- **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 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.
- **Concurrency**: [WSBL03a] 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.[1]
- **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.

[1] 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.

Table 2.3 Simulation results, H.264 decoder modeled in *ConcurrenC*

Filename	Boat.264				Coastguard.264			
# Macroblocks/frame	396				396			
# Frames	73 (2.43 s)				299 (9.97 s)			
# 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

- **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 [DGP+08].

Table 2.3 shows the simulation results of our H.264 decoder modeling in *ConcurrenC*. The model is simulated on a PC machine with Intel(R) Pentium(R) 4 CPU at 3.00 GHz. 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, four slices and eight 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 *ConcurrenC* is scalable. We can expect that it is possible to decode three of the test streams in real-time (bold times).

http://www.springer.com/978-3-319-08752-8

Out-of-order Parallel Discrete Event Simulation for Electronic System-level Design

Chen, W.

2015, XIX, 145 p. 51 illus., 41 illus. in color., Hardcover
ISBN: 978-3-319-08752-8