SAVE Project Report, Dept. of Computer and Information Science, Linköping University, Linköping, June 1999.

A Survey on Hardware/Software Codesign Representation Models

Luis Alejandro Cortés, Petru Eles and Zebo Peng
Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract

In hardware/software codesign, modeling is a very important issue. The model must capture the features of the system and describe its functionality. The design cycle must be based on formal representations so that the synthesis of a design from specification to implementation can be carried out systematically. Many models have been proposed for representing HW/SW systems. This report is the result of a survey on hardware/software codesign representation models. It relates the characteristics of several existing models and compares their properties. This work is encompassed in the SAVE project, which aims to study the specification and verification of heterogeneous electronic systems. The main objective of this survey is to explore the field of modeling of heterogeneous systems.

1. Introduction

Most modern electronic systems consist of dedicated hardware components and software running on specific platforms. Thus, complex systems might be composed of several standard processors and several ASICs working closely for a given task.

Mixed HW/SW systems are not new. What has considerably grown in recent years is the trend toward methodologies that concurrently apply design techniques from different areas to develop mixed digital systems. When hardware is tuned to its software applications, and vice versa, during the design process, it is possible to exploit the capabilities of such heterogeneous systems. HW/SW Codesign is "the system design process that combines the hardware and software perspectives from the earliest stages to exploit design flexibility and efficient allocation of functions" [Fra91].

The concurrent design of hardware and software has shown to be advantageous as long as HW and SW are considered as a whole instead of independent entities. Although benefits of hardware and software working together are evident, complex systems design involving both HW and SW is a non-trivial task. Designers have to manage complexity and heterogeneity, and they have multiple choices and possible implementations, which is the same reason that makes the design flexible. These are some of the new challenges for digital designers due to the interaction of different kinds of system philosophies.

Today the electronic market demands high-performance and low-cost products. Both performance and cost are essential to commercial competitiveness. Thus, the chip industry has faced two major challenges in order to satisfy the consumer needs: the increase in system complexity and the reduction in design times. High funcionality on a single chip and reduced time-to-market are goals that can be achieved through codesign methodologies exploiting the characteristics of heterogeneous systems. Besides performance and time-to-market, system reliability is a key aspect in the design of electronic systems. Here verification plays a key role.

It must also be pointed out that in order to devise systems that meet the performance, cost and reliability goals, the design process should be based on formal representations, that is, models that permit to carry out the design process from specification to implementation with a formal notation. Modeling is essential for design methods. It is important to note that a model is different from the language used to specify the system. The same model of computation can underlie several languages, and there exist some description languages that manage different models. Thus, model and language are two different concepts. Nonetheless, it is sometimes difficult to see a clear boundary between *model* and *language*. That is the case of e.g. the so-called *synchronous languages* used to model reactive systems.

1.1. Related Work

Many computational models have been proposed in the literature to represent digital systems. Those models encompass a broad range of styles, characteristics and application domains. Some works address the features of different models and languages, and compare their properties. Lee and Sangiovanni-Vicentelli [Lee96] propose the *tagged-signal model* as a framework for comparing characteristics from different models of computation. We consider such a representation as a metamodel which can not describe explicitly by itself a computational system. This “model” is used by Edwards *et al*. [Edw97] to examine and evaluate the properties of several representations employed in the design of embedded systems, including discrete event systems, communicating finite state machines, synchronous/reactive models, and dataflow process networks.

Narayan and Gajski [Nar93] discuss some issues related to the specification of embedded systems and examine the features of hardware description languages, namely VHDL, HardwareC, SDL, Statecharts, SpecCharts and CSP. In [Gaj97] some basic representation models are explained which include FSM, dataflow graph, and several variants and extensions of these. This work advocates certain qualities a model has to possess: formalism, completeness, comprehensiveness, and simplicity to describe systems.

2. Objective

The main purpose of this survey is to explore the field of modeling for heterogeneous systems. This is one of the first steps in the workplan of the SAVE Project (Specification and Verification of Heterogeneous Electronic Systems). This project aims at the development of a formal approach to specification, implementation and verification of heterogeneous electronic systems [SAV99].

Modeling hardware/software systems is a complex task because of their heterogeneity. Many approaches can be found in the literature. Some of them give rigorous mathematical treatment to the model with high level of formalism. This survey addresses some computational models used in hardware/software codesign. We review some known approaches to the modeling of heterogeneous systems. Special attention has been paid to those models that are appropriate for transformations.

3. Representation Models

A variety of models has been developed and used to represent heterogeneous systems. A model of computation should ideally comprehend concurrency, sequential behaviour and communication methods. Some models are intended for data-intensive systems while others are suitable for control-oriented applications, and few of them combine data/control as their application domain. Then the features can largely differ even though all representations presented in this survey are computational models for heterogeneous hardware/software systems.

In order to be consequent with the comparison of different models, the word *synchronous*—which has distinct meanings in different communities—must be first defined. In the following discussion, except when referring to *reactive/synchronous models*, we will use the term synchronous to mean systems with a global clock which synchronizes all the elements. In the frame of reactive representations, *synchronous* is used in a completely different sense meaning instantaneous reaction.

In the following we present the main representation models that have been utilized in codesign. We group them according to some characteristics they have in common. We do not pretend to be exhaustive in this survey but address some representative computational models used to depict heterogeneous electronic systems. In this section we briefly describe the key aspects of the models here studied and in next section we compare their most important features. Several codesign representations are derived from well-known models (e.g. FSMs or Petri nets), so a short description of significant characteristics of each group is done at the beginning of each subsection.

3.1. Finite State Machines

The classical *FSM* representation is maybe the most well-known model for describing control systems. This model consists of a set of states, a set of inputs, a set of outputs, a function which defines the outputs in terms of inputs and states, and a next-state function. One of its disadvantages is the exponential growth of the number of the states as the system complexity rises. Finite state machines do not allow concurrency of states and they are flat, that is to say, no hierarchical constructions are allowed. Besides, a small variation in the specification might produce a large change of the automata. In consequence FSMs are not appropriate for modeling practical systems. A number of extensions and variations of the typical FSM model have been suggested attempting to overcome the weaknesses of the FSM representation.

SOLAR is a design representation for high-level concepts in control-flow dominated systems and it is mainly suited for synthesis purposes [Jer95]. This representation is based on the extended-FSM model which might describe hierarchy and concurrency, and uses an EDIF-like syntax to model systems as a set of communicating FSMs. Systems are depicted by communicating *Design Units*. Design Units in SOLAR (system-level processes) communicate through either the classical concept of wired ports or the *Channel Unit* which supplies primitives that hide the protocol and thus its details. Each Design Unit is specified as a combination of states (FSMs), using the basic construct *StateTable*. As Design Units may comprise other units and communication operators, the computational model admits any level of hierarchy. The model provides an intermediate format that allows hardware/software designs at the system-level to be represented in such a way that they can be synthesized.

Codesign Finite State Machine (*CFSM*) is a formal model for hardware/software codesign proposed by Chiodo *et al*. [Chi93]. It is based on FSMs and is intended for control-oriented systems with relatively low algorithmic complexity. Even though its semantics is closely related to standard FSMs, the model can be used to represent both hardware and software. The communication primitive between

CFSMs, called *event*, is the basic entity which characterizes the behavior of the system to be modeled. The broadcast communication uses events with non-zero propagation time, where the sender does not wait for acknowledgment. The behavior of the system is defined as sequences of events that can be observed when interact with the environment. Due to its low level, CFSMs are used as intermediate representation which high-level languages are directly mapped into.

Even though Harel's *statecharts* [Har87] could be considered as an extension of traditional state diagrams, this computational model is mainly appropriate for large and complex reactive systems with high degree of concurrency. Statecharts have the structure of finite-state automata enhanced with three important features: hierarchy, concurrency and broadcast communication. The so-called *OR/AND* decomposition of states allows the first two characteristics. At some level of hierarchy, a particular state can be composed by substates which means that being in the higher-level state is interpreted as being in one of its several substates. Concurrent states are permitted in this model and some of the parallel states can similarly be seen as a single state using the property of hierarchy. The communication mechanism in statecharts is instantaneous broadcast, where the receiver proceeds immediately in response to the sender message. A statechart might contain instructions to carry out activities (computations or processes) either describing the states themselves or appending information on the label of transitions.

3.2. Discrete-Event Systems

A *discrete-event system* can be defined as a discrete-state event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time [Cas93]. An *event* is an instantaneous action that causes transitions from one discrete state to another. The interaction between computational tasks (processes) is accomplished by *signals*. In the discrete-event model of computation, a signal is a set of atomic events that occur in some instant of physical time. Thus, each event has a value and is marked with a time stamp. The events are sorted by time label and they are analyzed in chronological order. Signals allow the communication between processes that receive/send events. Lee gives a formal mathematical description of such systems [Lee98]. This framework advocates formal semantics for this model in order to maintain a consistent view of time in a system. In this approach, the mathematical formalism deals, among many other aspects of its semantics, with solutions to the problem of simultaneous events and zero-delay feedback loops, so the representation becomes deterministic. Though this kind of representation is useful for real-time systems, the principal disadvantage of discrete-event modeling is its cost: It is computationally expensive because it is necessary to sort globally all the events according with their time of occurrence.

3.3. Petri Nets

Modeling of systems using *Petri nets* has been applied widely in many fields of science. In the classical approach, a Petri net is composed of four basic elements: a set of *places*, a set of *transitions*, an *input function* that maps transitions to places, and an *output function* which is also a mapping from transitions to places. The mathematical formalism developed over many years, which defines its structure and firing rules, has made Petri nets a well-understood modeling tool [Pet81]. Two important intrinsic features of Petri nets are its concurrency and asynchronous nature. The former characteristic (parallelism) signifies that events may occur independently if they are enabled. The asynchronous property shows that there is no inherent clock mechanism for firing transitions. However, some communities claim that its drawbacks are the lack of hierarchical decomposition and the deficient expressiveness for computations.

Dittrich [Dit95] uses *Hierarchical Petri Nets* (*HPNs*) as a representation model. The main motivation behind this model is the difficulty for specifying and understanding Petri nets graphs of complex systems by flat representations. HPNs inherit most important properties of Petri nets, including concurrency and asynchrony. A system is modeled by bipartite directed graphs with inscriptions on the edges and the nodes. There are two kind of nodes: *transition* nodes represent active components (functions) whereas *place* nodes describe passive elements (data or conditions). The approach shows that some variations of Petri nets are appropriate for depicting the behavior of complex systems making use of hierarchically represented descriptions.

Stoy [Sto95] presents a modeling technique for hardware/software systems, based on an extended timed Petri nets (*ETPN*) notation [Pen94]. This Petri net representation, called *PURE*, is founded on a parallel model with data-control notation and provides timing information. PURE consists of two different, but closely related, parts: control unit and computational/data part. In this approach, timed Petri nets with restricted transition rules are used to represent control flow in both hardware and software. Hardware and software operations are represented by *datapaths* and *instruction dependence graphs* respectively. Communication between modules is modeled by pairs of I/O operations which form rendezvous points. As in classical Petri nets, PURE provides a concurrent and asynchronous description of the control. This representation allows to capture hardware and software in a consistent way, so it can be utilized during the synthesis process taking advantage of the efficient movement of funcionality between hardware and software domains.

3.4. Dataflow Graphs

Dataflow graphs are quite popular in modeling data-dominated systems. Computationally intensive systems—complex transformation and/or considerable transportation of data—might be conveniently represented by a directed graph where the *nodes* describe computations and the *arcs* represent the order in which the computations are performed. In this model all computations are executed only when the required operands are available. Another characteristic is that operations (nodes) behave as functions without side effects. As a result, computations may be performed either sequentially or concurrently. Nonetheless, the conventional dataflow graph model is inadequate for representing the control unit of systems. Many variations of dataflow models have been used to represent heterogeneous systems.

Dataflow process networks is a model of computation to be used in signal processing systems [Lee95]. Programs are specified by directed graphs where nodes (*actors*) represent computations and arcs (*streams*) represent sequences of data. Processing is done in series of iterated firings in which an actor transforms input data into output ones. Dataflow actors have firing rules to determine when they must be enabled and then execute a specific operation. In this kind of dataflow notation, communication of concurrent processes is done by unidirectional unbounded FIFO channels. The model also allows hierarchical representations of the graphs. Special cases of dataflow process networks, *synchronous data flow* (*SDF*) and *cyclo-static data flow* (*CSDF*), are compared in [Par95]. The difference between these models is that in SDF the actors consume and produce a fixed number of data tokens in each fire because of their static rules, while in CSDF actors have cyclically variable firing rules.

A *model graph* that integrates aspects of different models of computation is presented by Ziegenbein *et al.* [Zie98a]. The model graph is a directed bipartite graph composed of *process nodes*, *channel nodes* and a set of *edges*. The processes are enabled when the required input data is present, as in the classical dataflow model. Timing is considered associating a *latency time* with each process and each channel. This approach is based on basic constructs, augmented with annotations to capture details of the input representation. This internal representation allows scheduling and allocation of systems described by different models. Each input language is mapped into a specific set of annotations which are consistent in order to permit the basic purpose of scheduling and allocation. The model is expounded using an example which combines real-time operating systems (RTOS) and synchronous data flow (SDF) models. The processes of different models communicate via FIFO-ordered queues.

A directed, acyclic, polar graph—consisting of *processes*, and *single* and *conditional edges*—is used in [Ele98] as model for the system representation. The graph has two special nodes (*source* and *sink*) used to represent the first and last task. In this graph, each node represents a process which is assigned to one of the processing elements. These processing elements can be programmable processors, dedicated hardware components or buses. Thus, communication costs are considered by the so-called *communication processes*, mapped to allocated buses, which represent the connection that links processes assigned to different processors (programmable or hardware). The representation model allows each process to be characterized by an execution time and a guard which is the condition necessary to activate the task of that process. The *conditional process graph* model captures, at process level, both dataflow and flow of control.

3.5. Communicating Processes

In models derived from Hoare's *communicating sequential processes* (*CSP*)[1] [Hoa85], systems can be outlined as sets of elements (*processes*) that function independently and communicate with each other through unidirectional channels using a synchronizing protocol. Processes can be described in terms of *events* (atomic actions with zero duration). Time-consuming actions must be described using a pair of events. In CSP, synchronized communication means that the data transfer is done by a mechanism that ensures that the receiver process is in an adequate state to accept the information. Thus the processes stall until the message is transmitted. Thomas *et al.* use a set of independent, interacting, sequential processes, derived from CSP, as a model for representing the behavior of mixed hardware/software systems [Tho93]. These processes in the model correspond to hardware or software computations in the system to be represented. The model is mainly suited for system-level simulation and synthesis. It provides transformation capabilities that allow generation and evaluation of some design alternatives. The communication is performed through channels but, unlike in CSP, there exist additional primitives that permit unbuffered transfer and synchronization without data. Some of these communication primitives capture the interaction of hardware and software processes, and classify the synchronization and data transfer associated with each interaction. The model considers the shared memory itself as a process, so other processes can reach it through the channels.

3.6. Synchronous[2]/Reactive Models

Reactive systems are those which interact continuously with their environment. The so-called *synchronous model* for reactive systems is a representation for real-time systems. In this computational model, the output responses are produced simultaneously with the input stimuli. From this point of view, there is no observable time delay in the reaction. The advantage of this assumption (called the synchrony hypothesis) is that such systems are easier to describe and analyze. In [Ben91] an approach to synchronous modeling for reactive real-time systems is presented, and two styles of such a representation are stated under the names *multiple clocked recurrent systems* (*MCRSs*) and *state based formalisms*. The former style is more convenient for data predominant systems whereas the state based formalism is suitable for systems where flow of control is prevalent. The so-called *synchronous languages* implement this model in one of the above mentioned styles.

1. Even though actually there exists no restriction that processes must be sequential, the name remains.
2. Note that in this section, and when referring to reactive models, the term *synchronous* means zero reaction time, unlike the other computational models where synchronous means clocked.

ESTEREL is a language for programming reactive systems under the synchronous model of computation [Bou91]. The language permits to describe concurrent and sequential statements in synchronous systems. In this modeling approach, reactions are instantaneous and outputs are produced simultaneously. The synchronized modules communicate through signals (their status is present or absent). The communication mechanism is instantaneous broadcast, so all statements see the same value and status for any signal. Other significant feature of ESTEREL is its determinism. The language is adequate for systems that react to external stimuli from their environment and have control orientation.

On the other hand, *LUSTRE* [Hal91] is a language that adopts a data-oriented style. It is a declarative language which follows a synchronous dataflow approach and consents multirate clocking. The semantics of the language permits easily to define mathematical equation systems. In LUSTRE, as in any other synchronous language, interprocess communication is performed by instantaneous broadcast.

Another computational model based on the synchrony hypothesis is presented in [San99a]. The system is modeled using concurrent processes which are activated by events. In this model a signal is a set of events. The response and input signal of a process are synchronous. This means that each process has zero execution time. As each event is marked with a tag, events are totally ordered. Communication is done in terms of a simple data flow mechanism. In this approach, a system is specified and described in Haskell, a purely functional programming language.

4. Comparison of Models

Table 1 shows the chart which permits to evaluate the characteristics of the computational models described above. This table summarizes and also complements the various features of those models we have discussed. One of the most relevant traits a model must hold is its capability to represent concurrency as well as sequential behavior. Every model presented in this survey supports both qualities and then we omit the respective columns in Table 1.

The characteristics outlined in Table 1 correspond to: main application (which kind of systems are fairly represented by the model), timing (what timing aspects are considered in the model), clock mechanism (if the model is synchronous or asynchronous), communication method (how the different entities of the model communicate with each other), hierarchy (if the representation supports hierarchical structures), non-determinism (if the computational model allows a non-deterministic abstraction of the system behavior), and mathematical formalism (if a formal mathematical theory has been developed defining clearly the semantics of the model of computation).

5. Conclusions

This report describes the features of several codesign representation models. Most of them are well suited for data or control-oriented systems but few representations support widely both. There is predominance of asynchronous computational models in the sense that transitions are not driven by a clock signal. Even though timing is an important subject in modern electronic systems, many of the related models do not have an explicit notion of time. The majority of representations is based on a strong mathematical semantics, which makes them amenable for formal verification. As stated before, all models are capable of handling sequential and concurrent behavior, which is due to the nature of such software/hardware systems.

It is worth to point out that many different models coexist in the scenario of hardware/software codesign. These computational models have distinct features and support diverse kinds of applications. Consequently, the selection of an appropriate model for a given application must be based on the characteristics of this application.

Table 1.
Computational Models used in Hardware/Software Codesign

	Main Application	Timing	Clock Mechanism	Communication Method	Hierarchy	Non-Determinism	Math. Formalism
SOLAR [Jer95]	Control oriented	No explicit timing	Synchronous	Remote procedure call	Yes	No	No
CFSMs [Chi93]	Control oriented	Events with time stamp	Asynchronous	Events broadcasting	Yes	Yes	Yes
Statecharts [Har87]	Reactive	Timeout: min/max time spent in a state	Synchronous	Instantaneous broadcast	Yes	Yes	Yes
Discrete-Event [Lee98]	Real-time	Globally sorted events with time tag	Asynchronous	Wired signals	No	No	Yes
HPNs [Dit95]	Large-scale distributed	No explicit timing. Just order of transitions	Asynchronous	—	Yes	Yes	No
PURE [Sto95]	Control and data oriented	Places with time interval	Asynchronous	Rendezvous points	No	No	Yes
Dataflow Process Networks [Lee95]	Signal processing	No explicit timing	Asynchronous	Unbounded FIFO channels	Yes	No	Yes
Model Graph [Zie98]	Real-time	Latency time for processes and channels	Asynchronous	FIFO-ordered queues	No	Yes	Yes
Conditional Process Graph [Ele98]	Data and control oriented	All processes with execution time	Asynchronous	Communication processes	No	No	No
Communicating Processes [Tho93]	Data oriented	Atomic events along line of time	Asynchronous	Unidirectional channels	Yes	Yes	No
Synchronous (State Based) [Bou91]	Reactive (control oriented)	No explicit timing	Synchronous	Instantaneous broadcast	Yes	No	Yes
Synchronous (MCRSs) [Hal91]	Reactive (data oriented)	No explicit timing	Synchronous	Instantaneous broadcast	No	No	Yes
Synchronous Processes [San99a]	Reactive (data and control oriented)	Events with tag	Synchronous	Data flow mechanism	—	—	No

References

[Ben91] A. Benveniste and G. Berry, “The Synchronous Approach to Reactive and Real-Time Systems,” in *Proc. IEEE*, vol. 79, pp. 1270-1282, Sept. 1991.

[Ben95] T. Ben Ismail and A. Jerraya, “Synthesis Steps and Design Models for Codesign,” in *IEEE Computer*, vol. 28, pp. 44-52, Feb. 1995.

[Ber92] G. Berry and G. Gonthier, “The ESTEREL synchronous programming language: design, semantics, implementation,” in *Science of Computer Programming*, vol. 19, pp. 83-152, Nov. 1992.

[Bou91] F. Boussinot and R. de Simone, “The ESTEREL Language,” in *Proc. IEEE*, vol. 79, pp. 1293-1304, Sept. 1991.

[Cas93] C. G. Cassandras, *Discrete Event Systems: Modeling and Performance Analysis*. Boston, MA: Aksen Associates, 1993.

[Chi93] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vicentelli, “A Formal Specification Model for Hardware/Software Codesign,” Technical Report UCB/ERL M93/48, Dept. EECS, University of California, Berkeley, June 1993.

[Chi94] M. Chiodo, P. Giusto, A. Jurecska, H. Hsieh, A. Sangiovanni-Vicentelli, and L. Lavagno, “Hardware-Software Codesign of Embedded Systems,” in *IEEE Micro*, vol. 14, pp. 26-36, Aug. 1994.

[Dit95] G. Dittrich, “Modeling of Complex Systems Using Hierarchical Petri Nets,” in *Codesign: Computer-Aided Software/Hardware Engineering*, J. Rozenblit and K. Buchenrieder, Eds. Piscataway, NJ: IEEE Press, 1995, pp. 128-144.

[Edw97] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vicentelli, “Design of Embedded Systems: Formal Models, Validation, and Synthesis,” in *Proc. IEEE*, vol. 85, pp. 366-390, March 1997.

[Ele98] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, and P. Pop, “Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems,” in *Proc. DATE Conference*, 1998, pp. 132-138.

[Fra91] D. W. Frank and M. Purvis, “Hardware/Software CoDesign: A Perspective,” in *Proc. 13th Intl. Conference on Software Engineering*, 1991, pp. 344-352.

[Gaj97] D. D. Gajski, J. Zhu, and R. Dömer, “Essential Issues in Codesign,” in *Hardware/Software Co-Design: Principles and Practice*, J. Staunstrup and W. Wolf, Eds. Amsterdam: Kluwer, 1997, pp. 1-45.

[Hal91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, “The Synchronous Data Flow Programming Language LUSTRE,” in *Proc. IEEE*, vol. 79, pp. 1305-1320, Sept. 1991.

[Har87] D. Harel, “Statecharts: A Visual Formalism for Complex Systems,” in *Science of Computer Programming*, vol. 8, pp. 231-274, June 1987.

[Har90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot, “STATEMATE: A Working Environment for the Development of Complex Reactive Systems,” in *IEEE Trans. of Software Engineering*, vol. 16, pp. 403-414, April 1990.

[Hoa85] C. A. R. Hoare, *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[Jer95] A. Jerraya and K. O’Brien, “SOLAR: An Intermediate Format for System-Level Modeling and Synthesis,” in *Codesign: Computer-Aided Software/Hardware Engineering*, J. Rozenblit and K. Buchenrieder, Eds. Piscataway, NJ: IEEE Press, 1995, pp. 145-175.

[Lee95] E. A. Lee and T. Parks, “Dataflow Process Networks,” in *Proc. IEEE*, vol. 83, pp. 773-799, May 1995.

[Lee96] E. A. Lee and A. Sangiovanni-Vicentelli, “Comparing Models of Computation,” in *Proc. ICCAD*, 1996, pp. 234-241.

[Lee98] E. A. Lee, “Modeling Concurrent Real-Time Processes using Discrete Events,” Technical Report UCB/ERL M98/7, Dept. EECS, University of California, Berkeley, March 1998.

[Nar93] S. Narayan and D. D. Gajski, “Features Supporting System-Level Specification in HDLs,” in *Proc. EURO-DAC*, 1993, pp. 540-545.

[Par95] T. Parks, J. L. Pino, and E. A. Lee, “A Comparison of Synchronous and Cyclo-Static Dataflow,” in *Proc. 29th Asilomar Conference on Signals, Systems and Computers*, 1995, pp. 204-210.

[Pen94] Z. Peng and K. Kuchcinski, “Automated Transformation of Algorithms into Register-Transfer Level Implementations,” in *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 13, pp. 150-166, Feb. 1994.

[Pet81] J. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[Pop99] P. Pop, P. Eles, and Z. Peng, “An Improved Scheduling Technique for Time-Triggered Embedded Systems,” to appear in *Proc. EUROMICRO*, 1999.

[San99a] I. Sander and A. Jantsch, “Formal System Design Based on the Synchrony Hypothesis, Functional Models, and Skeletons,” in *Proc. VLSI Design*, 1999, pp. 318-323.

[[San99b] I. Sander and A. Jantsch, “System Synthesis Utilizing a Layered Functional Model,” in *Proc. CODES*, 1999, pp. 136-140.

[SAV99] “SAVE: Specification and Verification of Heterogeneous Electronic Systems,” Project Plan, March 1999.

[Sto94] E. Stoy and Z. Peng, “An Integrated Modelling Technique for Hardware/Software Systems,” in *Proc. ISCAS*, 1994, pp. 399-402.

[Sto95] E. Stoy, “A Petri Net Based Unified Representation for Hardware/Software Co-Design,” Licentiate Thesis, Dept. of Computer and Information Science, Linköping University, Linköping, 1995.

[Tho93] D. Thomas, J. Adams, and H. Schmit, “A Model and Methodology for Hardware-Software Codesign,” in *IEEE Design & Test of Computer*, vol. 10, pp. 6-15, Sept. 1993.

[Zie98a] D. Ziegenbein, R. Ernst, K. Ritcher, J. Teich, and L. Thiele, “Combining Multiple Models of Computation for Scheduling and Allocation,” in *Proc. CODES/CASHE*, 1998, pp. 9-13.

[Zie98b] D. Ziegenbein, K. Ritcher, R. Ernst, J. Teich, and L. Thiele, “Representation of Process Mode Correlation for Scheduling,” in *Proc. ICCAD*, 1998, pp. 54-61.