

# Modeling and Simulation of Heterogeneous Real-Time Systems Based on a Deterministic Discrete Event Model

J. Teich, L. Thiele \*

E. A. Lee

Computer Eng. and Comm. Networks Lab  
Swiss Federal Institute of Technology (ETH)  
CH-8092 Zürich, Switzerland  
email: {teich,thiele}@tik.ethz.ch

Department of EECS  
University of California at Berkeley  
Berkeley, CA 94720  
email: eal@eecs.berkeley.edu

## Abstract

*In this paper, an approach to system-level modeling and simulation of a class of heterogeneous real-time systems the timing behaviour of which can be modeled by deterministic discrete event systems is described. Examples of systems we consider are self-timed systems, synchronously clocked systems, and mixed asynchronous/synchronous systems. Our model is based on several extensions to the model of timed marked graphs. Basically, we augment this model by adding new schedule constraints such that we can express simultaneity, synchronicity, finite buffering as well as arbitrary combinations of min- and max-constraints. We prove that these extensions allow efficient timing analysis and we show how to simulate realistic systems using the Ptolemy [3] design system.*

## 1 Introduction

Whereas synthesis tools on lower levels of system design are fairly mature, there are still many open problems on higher levels of system design and integration. Typical problems are the synthesis of interfaces, hw/sw-codesign and co-simulation. The need for heterogeneous simulation and synthesis tools for different design styles (e.g., asynchronous versus synchronous computation and communication, hardware versus software, analog versus digital) is of utmost importance in order to obtain performance figures early. In case of real-time systems, exact timing and throughput estimation is one of the major requirements. A model for heterogeneous real-time designs should basically satisfy the following requirements:

- simplicity

- major focus on real-time constraints, concurrency, and synchronization
- ease of analysis (existence of efficient algorithms for analysis)
- versatility and extendibility

There exist many computational models for describing real-time systems, e.g., petri net based models [11, 4, 12], CSP [7], hierarchical finite state machines [6, 5], and block diagram oriented languages [9, 2], to name a few. A lot of research has been done to map a certain model to a software target e.g., a microcontroller, a DSP processor, a general purpose uniprocessor or a multiprocessor architecture, see e.g., [9]. Indeed, many results are available in this area on finding schedules (e.g., blocked, non-blocked, or cyclostatic [2]) together with an assignment of given computations to computing resources. Here, we are interested in modeling the performance of *dedicated systems* where assignment is given implicitly by dedicated resources. We are not primarily interested in finding a schedule to satisfy a certain throughput rate or some code size or communication requirements, but to analyze the fastest evolving schedules (e.g., *self schedules*) for a given assignment and communication style between system components. Such an analysis might be helpful, e.g., in obtaining performance figures that can be used for decision making in a system-level design optimization loop. Our model should be able to take into account relevant timing aspects like latency, throughput rates, and relevant aspects of communication including different communication styles, finite buffering, and interface constraints, i.e., timing constraints (e.g., min-, max- constraints and relative timing constraints). Hence, we are seeking to define a model that allows efficient timing analysis of dedicated real-time systems.

---

\*Part of this work was performed during a stay of the first author as a postdoctoral fellow in the Department of EECS at UC Berkeley in the Ptolemy project.

## 2 Basic computational model

### 2.1 Marked graphs and their unfolding

First, we introduce some notation and recall the computational model associated with marked graphs because this is the basis for the extensions described in this paper. For more details, see e.g., [11, 1] and the references therein. In the following, let  $\mathbf{N}$ ,  $\mathbf{Z}$ , and  $\mathbf{R}$  denote the set of natural numbers, integers, and real numbers, respectively.

**Definition 1 (Marked Graph)** *A timed marked graph  $G = (V, A, d, h)$  denotes a directed graph with*

- *nodes  $V = \{v_1, v_2, \dots, v_{|V|}\}$ ,*
- *arcs  $A = \{a_1, a_2, \dots, a_{|A|}\}$ , where any arc is an ordered pair of nodes  $a_p = (v_i, v_j)$ ,*
- *a distance function  $d : A \rightarrow \mathbf{Z}_{\geq 0}$  and a weight function  $h : A \rightarrow \mathbf{R}_{\geq 0}$ . We use  $d_{ij}$  and  $h_{ij}$  as shorthand notations for  $d(v_i, v_j)$  and  $h(v_i, v_j)$  associated with edge  $(v_i, v_j)$ .*

A node of  $G$  represents a computation module. Each arc  $a_p = (v_i, v_j)$  represents a queue of data directed from the computation module assigned to  $v_i$  to the computation module assigned to  $v_j$ . Thus, the computation module  $v_i$  places the results of its calculation onto arc  $a_p$  and the data on  $a_p$  are available as inputs to the computation module  $v_j$ . The distances and weights in Definition 1 are interpreted as follows:

- $d_{ij}$  denotes the initial number of data items (also called *tokens*) on arc  $a_p = (v_i, v_j)$ .
- $h_{ij}$  denotes the (minimum) holding time of a token in the queue associated with arc  $(v_i, v_j)$ .

These quantities are related to the token game one can play: If a node  $v_i$  fires, then the first token in all queues ending in  $v_i$  are removed and simultaneously, one token enters all queues originating at  $v_i$ . Now, a node can fire only (but need not fire immediately) if it is enabled. A token must spend the holding time  $h_{ij}$  in the queue from  $v_i$  to  $v_j$  before contributing to the enabling of the downstream node  $v_j$ . The firing of a node can't take place before all tokens in the queues contribute to the enabling. Many other essentially equivalent notions of marked graphs are possible ([12, 11, 1]).

Now, the set of all evolutions and scheduling of a timed marked graph can be described by the (event-) unfolding.

**Definition 2 (Unfolding)** *The unfolding of a timed marked graph  $G = (V, A, d, h)$  is an infinite directed graph  $\Sigma(G) = (V_\Sigma, A_\Sigma, h_\Sigma)$  where*

1.  $V_\Sigma$  contains nodes  $v_i(k)$  for all  $v_i \in V$  and for all integers  $k > -\max\{d_{ij} : v_j \in V \text{ such that } (v_i, v_j) \in A\}$ ,
2.  $A_\Sigma = \{(v_i(k - d_{ij}), v_j(k)) : (v_i, v_j) \in A \wedge k \in \mathbf{Z}_{>0}\}$ ,
3. *to each arc in  $A_\Sigma$  there is assigned the weight of the corresponding arc in  $G$ , i.e.  $h_\Sigma : A_\Sigma \rightarrow \mathbf{R}$  with  $h_\Sigma(v_i(k - d_{ij}), v_j(k)) = h_{ij}$  for all  $(v_i(k - d_{ij}), v_j(k)) \in A_\Sigma$ .*

A node  $v_i(k)$  of the unfolding represents the  $k$ th firing of node  $v_i$  in the marked graph. An edge from  $v_i(k)$  to  $v_j(l)$  denotes the fact that the  $l$ th firing of node  $v_j$  can take place only after the  $k$ th firing of node  $v_i$ . The nodes  $v_i(k)$  for  $k \leq 0$  represent the initial conditions of the marked graph, i.e. the placement of  $d_{ij}$  tokens into the queue corresponding to arc  $(v_i, v_j)$ .

Now, we can describe the scheduling of a timed marked graph more precisely. An admissible schedule is defined as follows:

**Definition 3 (Admissible Schedule)** *An admissible schedule function  $\sigma : V_\Sigma \rightarrow \mathbf{R}_{\geq 0}$  satisfies:*

1.  $\sigma_i(k) = 0$  for all  $v_i(k) \in V_\Sigma$ ,  $k \leq 0$
2.  $\sigma_j(k) \geq \sigma_i(k - d_{ij}) + h_{ij}$  for all  $(v_i(k - d_{ij}), v_j(k)) \in A_\Sigma$

where  $\sigma(v_i(k))$  is written as  $\sigma_i(k)$ .

Here,  $\sigma_i(k)$  denotes the time when node  $v_i$  fires for the  $k$ th time. From the interpretation of a marked graph it should be obvious that this can take place only if the corresponding input tokens have been in the queues  $(v_i, v_j)$  for at least time  $h_{ij}$ . As these tokens originate from the  $(k - d_{ij})$ th firings of nodes  $v_i$ , an admissible schedule of node  $v_j$  satisfies  $\sigma_j(k) \geq \sigma_i(k - d_{ij}) + h_{ij}$ . The first condition in Definition 3 serves to consider the initial timing conditions of the token game. All initial tokens are placed into the queues at time 0.

## 3 Model requirements

The model of timed marked graphs is not rich enough to describe all relevant aspects of heterogeneous real-time systems. Its main modeling deficiencies are:

- *simultaneity:* It should be possible to describe systems in which subsystems are constrained to work simultaneously.
- *synchronicity:* In synchronous systems, changes of system states can only occur simultaneous to certain events, e.g., edges of clock signals. It

should be possible to model cases where all nodes are clocked (= globally asynchronous/locally synchronous systems), some nodes are clocked (= mixed asynchronous/synchronous systems [13]), or it should be possible to emulate synchronously clocked systems as special cases.

- *min- and max-constraints*: In the domain of level-sensitive circuits, nodes are enabled for computation when the first signal change at its inputs arrives (= min-node) rather than after a signal change has occurred on all inputs (= max-nodes). Moreover, one would like to describe and simulate a system having min- as well as max-nodes.
- *finite buffering*: Limited buffering capacities will be modeled by introduction of reverse arcs in our graph model.
- *multi-rate-constraints*: It should be possible to describe different firing rates of nodes. For example, in digital signal processing, some nodes may consume or produce more than one output event per computation.

In this section, we will show how the timed marked graph model can be extended to cope with these constraints.

### 3.1 Modeling of self-timed systems

In a self-timed system with deterministic computation times, computation can be modeled by the marked graph model as introduced in the last section. The holding times  $h_{ij}$  therefore model the raw *computation time*  $c_j$  of the receiving node  $v_j$  plus a *communication time*  $w_{ij}$ , therefore  $h_{ij} = c_j + w_{ij}$ . Finite buffering capacities can be modeled by introducing reverse arcs and associating the communication time to those arcs (see, e.g., Fig. 3a, where arc  $(v_2, v_1)$  serves to model the constraint of a buffer size 3 along arc  $(v_1, v_2)$ ).

Next, we would like to consider synchronicity constraints. A typical case where such constraints need to be considered are synchronously clocked systems. If there is such a constraint on a node in our marked graph, we will call it a *synchronous node* in the following, otherwise we call it an *asynchronous node*.

### 3.2 Modeling of mixed asynchronous–synchronous systems

In a mixed asynchronous–synchronous system, a synchronous node can only initiate and finish computation at the edges of clock signals. In the following, we define an extended marked graph model called MASS (mixed asynchronous-synchronous system [13])

to include synchronicity constraints of timing events, in particular to periodic clock signals.

**Definition 4 (MASS)** *A mixed asynchronous -synchronous system (MASS) denotes an extended marked graph  $G = (V, A, d, h, r, p)$ . The set of nodes  $V$  is partitioned into disjoint subsets  $V_A$  and  $V_S$ , corresponding to asynchronous and synchronous nodes, respectively. In addition, the function  $r : V \rightarrow \mathbf{N}$  assigns a clock period, the function  $p : V \rightarrow \mathbf{R}$  assigns a clock phase  $0 \leq p_i < r_i$  to each node  $v_i \in V$ . In the unfolding  $\Sigma(G)$ , to each synchronous node  $v_i(k) \in V_\Sigma$  there is assigned the clock period  $r_i$  and the clock phase  $p_i$ .*

**Example 1** *Consider the MASS in Fig. 1. Node  $v_2$  is a synchronous node with the local clock phase  $p_2 = 0.1$ . Let  $r_2 = 1$  (normalized to 1).*

Here, the time instances  $\sigma_j(k)$  when a synchronous node can complete an operation are constrained as follows:

$$(\sigma_j(k) - p_j) \equiv 0 \pmod{r_j} \quad (1)$$

This restriction is motivated by the fact that a synchronous module can deliver a value at its local clock ticks only. Therefore, the firing of the node is delayed until the next clock event. For asynchronous nodes, clock period and clock phases play no role. However, by letting  $r_i = 1$  and  $p_i = 0$  for all  $v_i \in V_A$ , we can consolidate the description of the behaviour of synchronous and asynchronous nodes.

As a result, the definitions of admissible schedules must be extended as follows:

**Definition 5 (Admissible Schedule)** *An admissible schedule function of a mixed asynchronous–synchronous system is a function  $\sigma : V_\Sigma \rightarrow \mathbf{R}_{\geq 0}$  that satisfies  $\sigma_i(k) = \sigma_i(k)^* + p_i$ , and  $h_{ij} = h_{ij}^* + p_j - p_i$  where*

1.  $\sigma_i(k)^* = 0$  for all  $v_i(k) \in V_\Sigma$ ,  $k \leq 0$
2.  $\sigma_j(k)^* \geq F_j(\sigma_i(k - d_{ij})^* + h_{ij}^*)$  for all  $(v_i(k - d_{ij}), v_j(k)) \in A_\Sigma$

The function  $F_j$  is given by

$$F_j(a) = \begin{cases} a & : v_j \in V_A \\ \left\lceil \frac{a}{r_j} \right\rceil r_j & : v_j \in V_S \end{cases}$$

In case of an asynchronous node, the same definition as in Definition 3 is obtained because  $\sigma_j(k)^* \geq F_j(\sigma_i(k - d_{ij})^* + h_{ij}^*)$  is equivalent to  $\sigma_j(k) \geq \sigma_i(k - d_{ij}) - p_i + h_{ij} + p_i - p_j = \sigma_i(k - d_{ij}) + h_{ij}$ . In a self-schedule of a synchronous node,  $\sigma_j(k)$  is divisible by

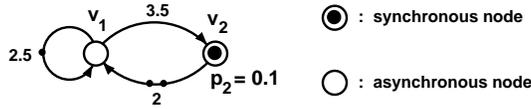


Figure 1: Example of a mixed asynchronous-synchronous system

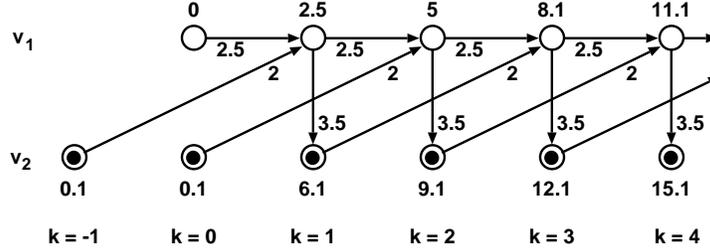


Figure 2: Unfolding of the system shown in Fig. 1

$r_j$  as desired. Moreover, the synchronous node fires at the *next* available clock tick. The stronger schedule condition

$$\sigma_j(k) \geq p_j + \left\lceil \frac{\sigma_i(k - d_{ij}) + h_{ij} - p_j}{r_j} \right\rceil r_j$$

is identical to Definition 5. Note that the term on the right hand side of this inequality is always greater than or equal to  $\sigma_i(k - d_{ij}) + h_{ij}$  but smaller than  $\sigma_i(k - d_{ij}) + h_{ij} + r_j$ , as desired.

**Example 2** *The same system as in Example 1 is considered. Its unfolding is shown in Fig. 2. Synchronous node  $v_2$ 's earliest firing time  $\sigma_2(2) = 9.1$  as  $\sigma_1(2) + 3.5 = 8.5$  has been rounded up to the next integer plus  $p_2 = 0.1$ .*

Looking at the definition of an unfolding, it is obvious that one possible maximal-rate schedule is obtained if a node fires as soon as it is enabled. This fact is elaborated in the following theorem.

**Theorem 1 (Free Schedule)** *The following conditions for the free schedule of a MASS are equivalent:*

1. *There is no admissible schedule with smaller firing times  $\sigma_i(k)$ .*
2.  $\sigma_j(k)^* = \max\{0, F_j(\sigma_i(k - d_{ij})^* + h_{ij}^*) : (v_i(k - d_{ij}), v_j(k)) \in A_\Sigma\}$

*The free schedule is a maximal-rate schedule.*

In [13], we have shown for the case of *single-clock rate* MASS, i.e., for the case  $r_j = 1 \forall v_j \in V_S$ , the maximal-rate can be determined accurately in polynomial time for special cases, e.g., for the cases  $V = V_A$  (self-timed systems),  $V = V_S$  (globally asynchronous, locally synchronous systems). In Definition 4, however, we consider the more general case when arbitrary combinations of asynchronous and synchronous nodes with possibly different clock periods.

### 3.3 Modeling of synchronous systems

Let a synchronous node stand for a *signal flow graph* (SFG) [10] in another level of hierarchy. The influence of retiming and pipelining can be easily incorporated into the MASS model by changing the computation times and node rates. However, there are also many possibilities to model a synchronous system by a MASS. For example, if only the sequence of operations is of concern, one may simply replace the registers by initial tokens and the combinational modules by asynchronous nodes of a marked graph. Another possibility is shown in Figs. 3b,c : The synchronous registers are replaced by synchronous nodes as shown in Fig. 3b. The combinational modules are modeled using asynchronous nodes whose holding times are the delays of the corresponding combinational modules, see Fig. 3c. The timing corresponds to that of the synchronous system if all accumulated delay times between registers are smaller than the clock period. A model for clock generation is shown in Fig. 3d: An extra clock node is responsible for the simultaneous

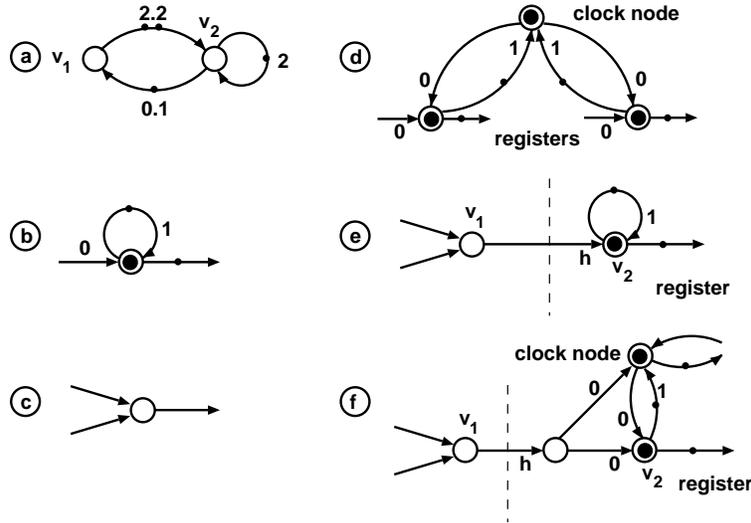


Figure 3: Modeling finite buffering (a), synchronous systems (b,c,d) and mixed asynchronous-synchronous systems (e,f) using the MASS model

transfer of token to *all* synchronous nodes which model the registers. The connection between the clock node and the synchronous nodes is bidirectional and corresponds to an interconnection with a queue length of one. This prevents the accumulation of clock tokens and guarantees that if the longest delay path between two registers is larger than the clock period, the period of the whole system is slowed down. Combinational modules are modeled as in Fig. 3c. A model for interfacing asynchronous and synchronous systems is shown in Fig. 3e. Let node  $v_1$  belong to an asynchronous subsystem whereas node  $v_2$  models the first register of a synchronous subsystem. This input register accepts tokens only at integral time instances. The holding time  $h$  models the communication time and the computation time of the synchronous node  $v_2$ , e.g., for executing the communication protocol. If the asynchronous part is faster than the synchronous one, tokens will accumulate in the queue ( $v_1, v_2$ ). This can be avoided by using finite length queues, see Fig. 3a. If the synchronous part is slower, the synchronous node  $v_2$  does not process a token every clock tick, but all other nodes in the synchronous subsystem do. This behavior can be implemented in hardware by sending a *token-bit* along with all data in the synchronous subsystem which signals that a register contains valid data. The clock operates continuously. Another possibility is shown in Fig. 3f. Here, the concept of a global clock node is used again. The incoming token is split

into one which represents the data and the other one which serves to enable the clock signal. This corresponds to realizations with *stoppable clocks*.

## 4 Simulation in Ptolemy

We want to simulate self-schedules of our extended times marked graphs (e.g., MASS) that can be characterized by a set of equations of the form:

$$\sigma_j(k) = F_j(\dots, \sigma_i(k - d_{ij}), \dots) \forall k \geq 0 \quad (2)$$

where  $F_j() \in \{\max(), \min(), \lceil \max() \rceil, \lfloor \min() \rfloor\}$ . Ptolemy [3] is a design environment for the design and simulation of heterogeneous systems. In one of its domains, the system contains a marked-graph scheduler (homogeneous SDF, see [9]). Now, to finish our idea of simulation, we define an isomorphic conventional (untimed) marked graph in Ptolemy with one node  $v_j$  for each equation and the node function given by  $F_j$ . Hence, the specific schedule constraints (synchronous, min, etc.) are hidden in the node functionality. Finally, any admissible schedule of this marked graph by definition has to wait for all arguments of the node function (tokens of incoming arcs) to exist. Hence, if  $v_j$  fires for the  $k$ th time, it computes  $\sigma_j(k)$  correctly, and outputs this token value to neighbor nodes:

**Theorem 2 (Correct Simulation)** *Any admissible marked graph schedule correctly computes the set of equations in (2).*

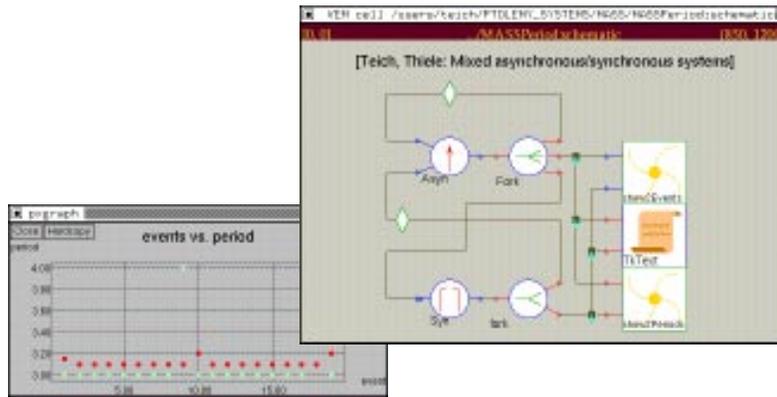


Figure 4: Example of a MASS simulated in Ptolemy

Hence, efficient simulation of our extended systems is obvious through the reduction to the computation of an arbitrary periodic schedule of a marked graph.

Fig. 4 shows the MASS of Fig. 1 in Ptolemy and a simulation window that displays the simulated node fire events of its self-schedule.

## 5 Extensions

Due to space requirements, we can only mention in a few words how our model can also cope with simultaneity and multi-rate constraints: The simultaneity of node firing times may be expressed by clustering nodes together into a supernode. Finally, we could model nodes consuming (or producing) different amounts of tokens by using the multi-rate SDF model [9] while keeping all mentioned timing constraints. Arbitrary combinations can be thought of, e.g., a *synchronous multi-rate min-node*. The formal analysis of such systems is subject of ongoing research. We claim that our ideas might be an important step towards timing simulation and system-level modeling of dedicated heterogeneous real-time systems.

## References

- [1] F. Baccelli, G. Cohen, G.J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. John Wiley, Sons, New York, 1992.
- [2] G. Bilsen, P. Wauters, M. Engels, R. Lauwereins, and J. Peperstraete. Development of a static load balancing tool. In *Proc. of the fourth Workshop on Parallel and Distr. Processing*, pages 179–194, Sofia, Bulgaria, 1993.
- [3] J. Buck, S. Ha, E.A. Lee, and D.G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *International Journal on Computer Simulation*, 4:155–182, 1994.
- [4] F. Commoner and A.W. Holt. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, 1971.
- [5] D. D. Gajski, F. Vahid, and S. Narayan. A system-design methodology: Executable-specification refinement. In *Proc. of the European Conference on Design Automation (EDAC)*, 1994.
- [6] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8, 1987.
- [7] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, NJ, 1985.
- [8] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, North Holland, 1974.
- [9] E.A. Lee and D.G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.
- [10] C.E. Leiserson, F.M. Rose, and J.B. Saxe. Optimizing synchronous circuitry by retiming. In *Proc. Third Caltech Conf. on VLSI*, pages 87–116, 1983.
- [11] J.L. Peterson. *Petri Net Theory and Modeling of Systems*. Prentice Hall, Reading, Mass., 1981.
- [12] R. Reiter. Scheduling parallel computations. *J. of the ACM*, 15:590–599, 1968.
- [13] J. Teich, S. Sriram, L. Thiele, and M. Martin. Performance Analysis of Mixed Asynchronous-Synchronous Systems. In *Proc. of the IEEE Int. Workshop on VLSI Signal Processing 94*, pages 103–112, October 1994.