Heterogeneous Reactive Systems Modeling and Correct-by-Construction Deployment*

Albert Benveniste[1], Luca P. Carloni[3], Paul Caspi[2], and
Alberto L. Sangiovanni-Vincentelli[3]

[1] Irisa/Inria, Campus de Beaulieu, 35042 Rennes cedex, France,
`Albert.Benveniste@irisa.fr`
`http://www.irisa.fr/sigma2/benveniste/`
[2] Verimag, Centre Equation, 2, rue de Vignate, F-38610 Gieres,
`Paul.Caspi@imag.fr`
`http://www.imag.fr/VERIMAG/PEOPLE/Paul.Caspi`
[3] U.C. Berkeley, Berkeley, CA 94720,
`{lcarloni,alberto}@eecs.berkeley.edu`
`http://www-cad.eecs.berkeley.edu/HomePages/{lcarloni,alberto}`

Abstract. We propose a mathematical framework to deal with the composition of heterogeneous reactive systems. Our theory allows to establish theorems, from which design techniques can be derived. We illustrate this by two cases: the deployment of synchronous designs over GALS architectures, and the deployment of synchronous designs over the so-called Loosely Time-Triggered Architectures.

1 Introduction

The *notion of time* has been a crucial aspect of electronic system design for years. Dealing with concurrency, time and causality has become increasingly difficult as the complexity of the design grows. The *synchronous programming model* has had major successes at the specification level because it provides a simpler way to access the power of concurrency in functional specification. Synchronous Languages like ESTEREL [8], LUSTRE [14], and SIGNAL [19], the STATECHARTS modeling methodology [15], and design environments like SIMULINK/STATEFLOW [23] all benefit from the simplicity of the *synchronous assumption*, i.e.: (1) the system evolves through an infinite sequence of successive atomic reactions indexed by a *global logical clock*, (2) during a reaction each component computes new events for all its output signals based on the presence/absence of events computed in the previous reaction and, (3) the communication of events among components occur *instantaneously* between two successive reactions.

However, if the synchronous assumption simplifies system specification, the problem of deriving a correct physical implementation from it does remain. In

* This research was supported in part by the European Commission under the projects COLUMBUS, IST-2002-38314, and ARTIST, IST-2001-34820, by the NSF under the project ITR (CCR-0225610), and by the GSRC.

particular, difficulties arise when the target architecture for the embedded system has a distributed nature that badly matches the synchronous assumption because of large variance in computation and communication times and because of the difficulty of maintaining a global notion of time. This is increasingly the case for many important classes of embedded applications in avionics, industrial plants, and the automotive industry. Here, multiple processing elements operating at different clock frequencies are distributed on an extended area and connected via communication media such as busses (e.g., CAN for automotive applications, ARINC for avionics, and Ethernet for industrial automation) or serial links. Busses and serial links can, however, be carefully designed to comply with a notion of global synchronization as the family of Time-Triggered Architectures (TTA), introduced and promoted by H. Kopetz [17], testifies. A synchronous implementations must be conservative, forcing the clock to run as slow as the slowest computation/communication process (*worst-case approach*). The overhead implied by time-triggered architectures and synchronous implementations is often enough to convince designers to use asynchronous communication architectures such as the ones implemented by the busses mentioned above.

We argue that imposing an "homogeneous" design policy, such as the fully synchronous approach, on complex designs will be increasingly difficult. Heterogeneity will manifest itself at the component level where different models of computation may be used to represent component operation and, more frequently, at different levels of abstraction, where, for example, a synchronous-language specification of the design may be refined into a globally asynchronous locally synchronous (GALS) architecture.

In this paper, we provide a mathematical framework for the heterogeneous modeling of reactive and real-time systems to allow freedom of choice between different synchronization policies at different stages of the design. The focus of our framework is handling communication and coordination among heterogeneous processes in a mathematically sound way. Interesting work along similar lines has been the Ptolemy project [13,22], the MoBIES project [1], the Model-Integrated Computing (MIC) framework [16], and *Interface Theories* [12].

Our main contributions are a mathematical model for heterogeneous system built as a variation of the "Tagged-Signal Model" of Lee and Sangiovanni-Vincentelli [21] (in this paper, called the *LSV model*) and a set of theorems that support effective techniques to generate automatically correct-by-construction adaptors between designs formulated using different design policies. We illustrate these concepts with two applications that are of particular relevance for the design of embedded systems: the deployment of a synchronous design over a GALS architecture and over a so-called Loosely Time-Triggered Architecture (LTTA) [7]. The idea followed in these examples is to abstract away from the synchronous specifications the constraints among events of different signals due to the synchronous paradigm and, then, to map the "unconstrained" design into a different architecture characterized by a novel set of constraints among events. In doing so, we must make sure that, when we remap the design, the intended "behaviour" of the system is retained. To do so we introduce a formal notion of *semantic preserving* transformation. The constraints on coordination among processes are captured by using the “tags” in the LSV model and the transformations are handled with morphisms among tag sets. For more details, the reader is referred to the extended version [5] of this paper.

2 Tagged Systems and Heterogeneous Systems

In this section, we build on the Lee and Sangiovanni-Vincentelli (LSV) “Tagged-Signal Model” [21]. For reasons that will be clear in the sequel, we slightly deviate from the original LSV model.

2.1 Tagged Systems

Symbol $\mathbf{N}$ denotes the set of positive integers. $X \mapsto Y$ denotes the set of all maps having X as domain, and whose range is contained in Y. Also, if $(X, \leq_X)$ and $(Y, \leq_Y)$ are partial orders, a map $f \in X \mapsto Y$ is called *nondecreasing* if

$$\forall x, x' \in X : x \leq_X x' \Rightarrow \neg[f(x') <_Y f(x)] \tag{1}$$

Condition (1) expresses that a nondecreasing map cannot invert orders. However, it can add or remove some orders. Thus, nondecreasing is weaker than increasing.

Definitions. We assume an underlying partially ordered set $\mathcal{T}$ of *tags,* we denote by $\leq$ the partial oder on $\mathcal{T}$, and we write $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$. A *clock* is a nondecreasing map $h \in \mathbf{N} \mapsto \mathcal{T}$. Assume an underlying set $\mathcal{V}$ of variables, with domain D. For $V \subset \mathcal{V}$ finite, a *V-behaviour,* or simply behaviour, is an element:

$$\sigma \ \in \ V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D), \tag{2}$$

meaning that, for each $v \in V$, the n-th occurrence of v in behaviour σ has tag $\tau \in \mathcal{T}$ and value $x \in D$. For v a variable, the map $\sigma(v) \in \mathbf{N} \mapsto (\mathcal{T} \times D)$ is called a *signal.* For σ a behaviour, an *event* of σ is a tuple $(v, n, \tau, x) \in V \times \mathbf{N} \times \mathcal{T} \times D$ such that $\sigma(v)(n) = (\tau, x)$; thus we can regard behaviours as sets of events. We require that, for each $v \in V$, the 1st projection of the map $\sigma(v)$ (it is a map $\mathbf{N} \mapsto \mathcal{T}$) is nondecreasing. Thus it is a clock, we call it the *clock of v in σ.*

A *tagged system* is a triple $P = (V, \mathcal{T}, \Sigma)$, where V is a finite set of variables, $\mathcal{T}$ is a tag set, and Σ a set of V-behaviours. If $\mathcal{T}_1 = \mathcal{T}_2 =_{\mathrm{def}} \mathcal{T}$, the *parallel composition* of systems P_1 and P_2 is by intersection:

$$\begin{aligned} P_1 \,\|\, P_2 &=_{\mathrm{def}} (V_1 \cup V_2, \mathcal{T}, \Sigma_1 \wedge \Sigma_2), \text{ where} \\ \Sigma_1 \wedge \Sigma_2 &=_{\mathrm{def}} \left\{\sigma \,\middle|\, \sigma_{|V_i} \in \Sigma_i, i = 1, 2\right\}, \end{aligned} \tag{3}$$

and $\sigma_{|W}$ denotes the restriction of σ to a subset W of variables. The set $\mathcal{T}$ of tags can be adjusted to account for different classes of systems.

Synchronous Systems. To represent *synchronous systems* with our model, take for $\mathcal{T}$ a totally ordered set, and require that all clocks are *strictly increasing.* The tag index set $\mathcal{T}$ organizes behaviours into successive reactions, as explained next. Call *reaction* a maximal set of events of σ with identical τ. Since clocks are strictly increasing, no two events of the same reaction can have the same variable. Regard a behaviour as a sequence of global reactions: $\sigma = \sigma_1, \sigma_2, \ldots$, with tags $\tau_1, \tau_2, \ldots \in \mathcal{T}$. Thus $\mathcal{T}$ provides a global, logical time basis. Particular instances for $\mathcal{T}$ correspond to different views of synchronous systems:

– Taking $\mathcal{T} = \mathbf{N}$ means that we assume some basic logical clock (the identity map, from $\mathbf{N}$ to $\mathbf{N}$), which is global to all considered systems, and all clocks are sub-clocks of this basic one. This is a good model for closed systems.
– Now, take for $\mathcal{T}$ a totally ordered dense set—e.g., $\mathcal{T} = \mathbf{R}$, the set of real numbers, or $\mathbf{Q}$, the set of rational numbers. Then, for any given clock h, there exists another clock k whose range has empty intersection with the range of h. This models the fact that, for any given system, there exists another system that is working while the former is sleeping; this is a suitable model for open systems. In fact, adequate models for open systems are stuttering invariant systems we define next. Call *time change* any bijective and strictly increasing function $\rho\ : \mathcal{T} \mapsto \mathcal{T}$, and denote by $\mathsf{R}_{\mathcal{T}}$ the set of all time changes over $\mathcal{T}$. Then a synchronous system $P = (V, \mathcal{T}, \Sigma)$ is called *stuttering invariant* iff it is invariant under time change, i.e., for every behaviour $\sigma \in \Sigma$ and every time change $\rho \in \mathsf{R}_{\mathcal{T}}$, then $\sigma^\rho \in \Sigma$ holds, where

$$(v, n, \tau, x) \in \sigma^\rho \ \Leftrightarrow_{\text{def}}\ \sigma(v, n, \rho(\tau), x) \in \sigma. \tag{4}$$

Examples of stuttering invariant systems are the stallable processes of latency-insensitive design [9], where $\mathcal{T} = \mathbf{N}$.

Timed Synchronous Systems. Timed synchronous systems are synchronous systems in which physical time is available, in addition to the ordering of successive reactions. Note that events belonging to the same reaction may occur at different physical instants. For this case we take $\mathcal{T} = \mathcal{T}_{\text{synch}} \times \mathcal{T}_\varphi$, where $\mathcal{T}_{\text{synch}}$ indexes the reactions, and $\mathcal{T}_\varphi$ is the physical time basis (e.g., $\mathcal{T}_\varphi = \mathbf{R}$ for real-time).

Asynchronous Systems. The notion of asynchronicity is vague. Any system that is not synchronous could be called asynchronous, but we often want to restrict somewhat this notion to capture particular characteristics of a model. In this paper, we take a very liberal interpretation for an asynchronous system. If we interpret a tag set as a constraint on the coordination of different signals of a system and the integer $n \in \mathbf{N}$ as the basic constraint of the sequence of events of the behaviour of a variable, then the most “coordination unconstrained” system, the one with most degrees of freedom in terms of choice of coordination mechanism, could be considered an ideal asynchronous system. Then an asynchronous system corresponds to a model where the tag set does not give any information on the absolute or relative ordering of events. In more formal way, take $\mathcal{T} = \{.\}$, the trivial set consisting of a single element. Then, behaviours identify with elements $\sigma \in V \mapsto \mathbf{N} \mapsto D$.

Running example. This simple example will be used throughout the rest of the paper to illustrate our results and their implications. Let P and Q be two synchronous systems involving the same set of variables: b of type boolean, and x of type integer. Each system possesses only a single behaviour, shown on the right hand side of $P : \ldots$ and $Q : \ldots$, respectively. Each behaviour consists of a sequence of successive reactions, separated by vertical bars. Events sitting in the same reaction can be seen *aligned.* Each reaction consists of an assignment of values to a subset of the variables; a blank indicates the absence of the considered variable in the considered reaction.

$$P : \begin{array}{r|c|c|c|c|c|c|l} \hline b : & t & f & t & f & t & f & \ldots \\ x : & 1 & & 1 & & 1 & & \ldots \\ \hline \end{array} \;, \quad Q : \begin{array}{r|c|c|c|c|c|c|l} \hline b : & t & f & t & f & t & f & \ldots \\ x : & & 1 & & 1 & & 1 & \ldots \\ \hline \end{array}$$

One of the questions addressed in this paper is how to deploy a synchronous design on a less constrained architecture. The general strategy we follow is to eliminate first the constraints introduced by the synchronous assumption and then to map the resulting asynchronous system on a less constrained architecture that may range from asynchronous to relaxed versions of timed-triggered architectures. The *desynchronization* of a synchronous system like P or Q consists in (1) removing the synchronization barriers separating the successive reactions, and, then, (2) compressing the sequence of values for each variable, individually. This yields:

$$P_\alpha = Q_\alpha : \begin{array}{l} \hline b : t \; f \; t \; f \; t \; f \; \ldots \\ \hline x : 1 \; 1 \; 1 \; \ldots \\ \hline \end{array}$$

where the subscript $_\alpha$ refers to asynchrony. The reader may think that events having identical index for different variables are aligned, but this is not the case. In fact, as the absence of vertical bars in the diagram suggests, there is *no alignment* at all between events associated with different variables.

Regarding desynchronization, the following comments are in order. Note that $P \neq Q$ but $P_\alpha = Q_\alpha$. Next, the synchronous system R defined by $R = P \cup Q$, the nondeterministic choice between P and Q, possesses two behaviours. However, its desynchronization R_α equals P_α, and possesses only one behaviour.

Now, we use the proposed framework to derive formal models for P, Q, and P_α. For the synchronous systems P and Q, we take $\mathcal{T} = \mathbf{N}$ to index the successive reactions. P possesses a single behaviour (note the absence of x at tag $2n$):

$$\begin{array}{lll} \sigma(b)(2n-1) = (2n-1, t) & , & \sigma(b)(2n) = (2n, f) \\ \sigma(x)(n) \quad\;\; = (2n-1, 1) & & \end{array} \tag{5}$$

For Q, we have (note the difference):

$$\begin{array}{lll} \sigma(b)(2n-1) = (2n-1, t) & , & \sigma(b)(2n) = (2n, f) \\ & & \sigma(x)(n) \;\; = (2n, 1) \end{array} \tag{6}$$

For the asynchronous systems $P_\alpha = Q_\alpha$, we take $\mathcal{T} = \{.\}$, the trivial set with a single element. The reason is that we do not need any additional time stamping information. Thus, $P_\alpha = Q_\alpha$ possess a single behaviour:

$$\sigma_\alpha(b)(2n-1) = t, \sigma_\alpha(b)(2n) = f, \text{ and } \sigma_\alpha(x)(n) = 1. \tag{7}$$

Discussion. A proactive reader would immediately criticize the definition (2) of behaviours. Our definition uses two indexing mechanisms, namely $\mathbf{N}$, to order events in each individual signal, and $\mathcal{T}$, to order (time stamp) events globally across signal boundaries. Of course, since the events of a signal are totally ordered by their "time stamps", their local index n results redundant. The redundancy of the indexing is particularly visible in (5) and (6). The reason for introducing the redundancy is related to the different semantics of the two orders: one is intrinsic in the very notion of events of a signal, the other is related to the constraints on event ordering due to the synchronous assumption. Decoupling the two orders allows us to represent cleanly the desynchronization operation and the deployment on more general architectures. We refer to [5] for a more detail discussion with respect to the original LSV model.

2.2 Heterogeneous Systems

Assume a functional system specification using a synchronous approach, for subsequent deployment over a distributed asynchronous architecture (synchronous and asynchronous are considered in the sense of subsection 2.1). When we deploy the design on a different architecture, we must make sure that the original intent of the designer is maintained. This step is non trivial because the information on what is considered correct behaviour is captured in the synchronous specifications that we want to relax in the first place. We introduce the notion of semantic-preserving transformation to identify precisely what is a correct deployment. We present the idea with our running example:

Running example, cont'd. The synchronous parallel composition of P and Q, defined by intersection: $P \parallel Q =_{\text{def}} P \cap Q$, is empty. The reason is that P and Q disagree on where to put absences for the variable x. On the other hand, since $P_\alpha = Q_\alpha$, then $P_\alpha \parallel Q_\alpha =_{\text{def}} P_\alpha \cap Q_\alpha = P_\alpha = Q_\alpha \neq \emptyset$. Thus, for the pair (P, Q), desynchronization does not preserve the semantics of parallel composition, in any reasonable sense. $\diamond$

How to model that semantics is preserved when replacing the ideal synchronous broadcast by the actual asynchronous communication? In the case of deployment using the Ltta-protocol, we face the same issues, but with the occurrence of time as an additional component of tags. In fact, an elegant solution was proposed by Le Guernic and Talpin for the former GALS case [20]. We cast their approach in the framework of tagged systems and we generalize it.

Morphisms. For $\mathcal{T}, \mathcal{T}'$ two tag sets, call *morphism* a map $\rho : \mathcal{T} \mapsto \mathcal{T}'$ which is nondecreasing and surjective[1]. For $\rho : \mathcal{T} \mapsto \mathcal{T}'$ a morphism, and $\sigma \in V \mapsto \mathbf{N} \mapsto (\mathcal{T} \times D)$ a behaviour, replacing τ by $\rho(\tau)$ in σ defines a new behaviour having $\mathcal{T}'$ as tag set, we denote it by

$$\sigma_\rho, \text{ or by } \sigma{\circ}\rho. \tag{8}$$

Performing this for every behaviour of a tag system P yields the tag system

$$P_\rho. \tag{9}$$

For $\mathcal{T}_1 \xrightarrow{\rho_1} \mathcal{T} \xleftarrow{\rho_2} \mathcal{T}_2$ two morphisms, define:

$$\mathcal{T}_1 \;{}_{\rho_1}{\times}_{\rho_2}\; \mathcal{T}_2 =_{\text{def}} \{ (\tau_1, \tau_2) \mid \rho_1(\tau_1) = \rho_2(\tau_2) \}. \tag{10}$$

A case of interest is $\mathcal{T}_i = \mathcal{T}_i' \times \mathcal{T}, i = 1, 2$, and the $\mathcal{T}_i'$ are different. Then $\mathcal{T}_1 \;{}_{\rho_1}{\times}_{\rho_2}\; \mathcal{T}_2$ identifies with the product $\mathcal{T}_1' \times \mathcal{T} \times \mathcal{T}_2'$.

Desynchronization. For example, the *desynchronization* of synchronous systems is captured by the morphism $\alpha : \mathcal{T}_{\text{synch}} \mapsto \{.\}$, which erases all global timing information (see Equations (5,6), and (7)).

Heterogeneous Parallel Composition. In this subsection we define the composition of two tagged systems $P_i = (V_i, \mathcal{T}_i, \Sigma_i), i = 1, 2$, when $\mathcal{T}_1 \neq \mathcal{T}_2$. This definition is provided in two stages. For the first stage, we assume that $\mathcal{T}_1 = \mathcal{T}_2$. For σ_i a behaviour of P_i, say that (σ_1, σ_2) is a *unifiable* pair, written

$$\sigma_1 \bowtie \sigma_2 \text{ iff } \sigma_{1|V_1 \cap V_2} = \sigma_{2|V_1 \cap V_2}.$$

For $\sigma_1 \bowtie \sigma_2$, the *unification* of σ_1 and σ_2 is the behaviour $\sigma_1 \sqcup \sigma_2$ having $V_1 \cup V_2$ as set of variables, and such that:

$$(\sigma_1 \sqcup \sigma_2)_{|V_i} = \sigma_i, i = 1, 2.$$

Now, return to the case $\mathcal{T}_1 \neq \mathcal{T}_2$, and assume two morphisms $\mathcal{T}_1 \xrightarrow{\rho_1} \mathcal{T} \xleftarrow{\rho_2} \mathcal{T}_2$. Write:

$$\sigma_1 \;{}_{\rho_1}{\bowtie}_{\rho_2}\; \sigma_2 \text{ iff } \sigma_1{\circ}\rho_1 \bowtie \sigma_2{\circ}\rho_2. \tag{11}$$

For (σ_1, σ_2) a pair satisfying (11), define

$$\sigma_1 \;{}_{\rho_1}{\sqcup}_{\rho_2}\; \sigma_2 \tag{12}$$

as being the set of events $(v, n, (\tau_1, \tau_2), x)$ such that $\rho_1(\tau_1) = \rho_2(\tau_2) =_{\text{def}} \tau$ and (v, n, τ, x) is an event of $\sigma_1{\circ}\rho_1 \sqcup \sigma_2{\circ}\rho_2$. We are now ready to define the *heterogeneous conjunction* of Σ_1 and Σ_2 by:

$$\Sigma_1 \;{}_{\rho_1}{\wedge}_{\rho_2}\; \Sigma_2 =_{\text{def}} \{ \sigma_1 \;{}_{\rho_1}{\sqcup}_{\rho_2}\; \sigma_2 \mid \sigma_1 \;{}_{\rho_1}{\bowtie}_{\rho_2}\; \sigma_2 \} \tag{13}$$

[1] Strictly speaking, these are not morphisms of order structures. We use this word by abuse of terminology.

Finally, the *heterogeneous parallel composition* of P_1 and P_2 is defined by:

$$P_1 \ {}_{(\rho_1}\|_{\rho_2)} \ P_2 = (\, V_1 \cup V_2 \,,\, \mathcal{T}_1 \ {}_{\rho_1}\!\times_{\rho_2} \mathcal{T}_2 \,,\, \Sigma_1 \ {}_{\rho_1}\!\sqcup_{\rho_2} \Sigma_2 \,)\,. \tag{14}$$

We simply write ${}_{(\rho_1}\|$ instead of ${}_{(\rho_1}\|_{\rho_2)}$ when ρ_2 is the identity.

GALS and Hybrid Timed/Untimed Systems. To model the interaction of a synchronous system with its asynchronous environment, take the heterogeneous composition $P {}_{(\alpha}\| \ A$, where $P = (V, \mathcal{T}_{\text{synch}}, \Sigma)$ is a synchronous system, $A = (W, \{.\}, \Sigma')$ is an asynchronous model of the environment, and $\alpha : \mathcal{T}_{\text{synch}} \mapsto \{.\}$ is the trivial morphism, mapping synchrony to asynchrony (hence the special notation).

For GALS, take $\mathcal{T}_1 = \mathcal{T}_2 = \mathcal{T}_{\text{synch}}$, where $\mathcal{T}_{\text{synch}}$ is the tag set of synchronous systems. Then, take $\mathcal{T} = \{.\}$ is the tag set of asynchronous ones. Take $\alpha : \mathcal{T}_{\text{synch}} \mapsto \{.\}$, the trivial morphism. And consider $P_1 \ {}_{(\alpha}\|_{\alpha)} \ P_2$.

For timed/untimed systems, consider $P {}_{(\rho}\| \ Q$, where $P = (V, \mathcal{T}_{\text{synch}} \times \mathcal{T}_\varphi, \Sigma)$ is a synchronous timed system, $Q = (W, \mathcal{T}_{\text{synch}}, \Sigma')$ is a synchronous but untimed system, and $\rho : \mathcal{T}_{\text{synch}} \times \mathcal{T}_\varphi \mapsto \mathcal{T}_{\text{synch}}$ is the projection morphism.

3 Application to Correct Deployment

In this section we formalize the concept of semantics preserving and present a general result on correct-by-construction deployment.

3.1 Preserving Semantics: Formalization

We are given a pair $P_i = (V_i, \mathcal{T}_i, \Sigma_i), i = 1, 2$, such that $\mathcal{T}_1 = \mathcal{T}_2$, and a pair $\mathcal{T}_1 \xrightarrow{\rho} \mathcal{T} \xleftarrow{\rho} \mathcal{T}_2$ of identical morphisms. We can consider two semantics:

$$\begin{aligned} \textit{The strong semantics} &: \ P_1 \,\|\, P_2 \\ \textit{The weak semantics} &: \ P_1 \ {}_{(\rho}\|_{\rho)} \ P_2. \end{aligned}$$

We say that ρ is *semantics preserving* with respect to $P_1 \,\|\, P_2$ if

$$P_1 \ {}_{(\rho}\|_{\rho)} \ P_2 \ \equiv \ P_1 \,\|\, P_2. \tag{15}$$

Running example, cont'd. The reader can check the following as an exercise: $P \,\|\, Q = \emptyset$, and, as we already discussed, $P_\alpha \,\|\, Q_\alpha = P_\alpha$. Now we compute $P \ {}_{(\alpha}\|_{\alpha)} \ Q$. From (12) we get that, using obvious notations, (σ_P, σ_Q) is a pair of behaviours that are unifiable modulo desynchronization, i.e., $\sigma_P \ {}_{\alpha}\!\bowtie_{\alpha} \sigma_Q$. Then, unifying these yields the behaviour σ such that:

$$\forall n \in \mathbf{N} : \sigma(b)(n) = ((n, n), v_b) \text{ and } \sigma(x)(n) = ((2n-1, 2n), 1) \tag{16}$$

where $v_b = t$ if n is odd, and $v_b = f$ if n is even. In (16), the expression for $\sigma(b)(n)$ reveals that desynchronizing causes no distortion of logical time for b, since (n, n) attaches the same tag to the two behaviours for unification. On the other hand, the expression for $\sigma(x)(n)$ reveals that desynchronizing actually causes distortion of logical time for x, since $(2n-1, 2n)$ attaches different tags to the two behaviours for unification. Thus $P \,\|\, Q = \emptyset$, but $P \;_{(\alpha}\|_{\alpha)}\; Q$ consists of the single behaviour defined in (16). Hence, $P \;_{(\alpha}\|_{\alpha)}\; Q \not\equiv P \,\|\, Q$ in this case: semantics is not preserved. $\diamond$

3.2 A General Result on Correct Deployment

Here we analyse requirement (15). The following theorem holds (see (9) for the notation P_ρ used in this theorem):

Theorem 1. *The pair (P_1, P_2) satisfies condition (15) if it satisfies the following two conditions:*

$$\forall i \in \{1,2\} : (P_i)_\rho \text{ is in bijection with } P_i \tag{17}$$

$$(P_1 \;\|\; P_2)_\rho \;=\; (P_1)_\rho \;\|\; (P_2)_\rho \tag{18}$$

Comments. The primary application of this general theorem is when P and Q are synchronous systems, and $\rho = \alpha$ is the desynchronization morphism. This formalizes GALS deployment. Thus, Theorem 1 provides sufficient conditions to ensure correct GALS deployment.

Conditions (17) and (18) are not effective because they involve (infinite) behaviours. In [3,4], for GALS deployment, condition (17) was shown equivalent to some condition called *endochrony,* expressed in terms of the transition relation, not in terms of behaviours. Similarly, condition (18) was shown equivalent to some condition called *isochrony,* expressed in terms of the pair of transition relations, not in terms of pairs of sets of behaviours. Endochrony and isochrony are model checkable and synthesizable, at least for synchronous programs involving only finite data types (see [3,4] for a formal definition of these conditions).

In the same references, it was claimed that the two conditions (17) and (18) "mean" the preservation of semantics for a GALS deployment of a synchronous design. Several colleagues pointed to us that they did not see why this claim should be obvious. In the subsequent paper [2], an attempt was provided to fill this gap, with no real formalization, however. Theorem 1 provides the missing formal justification for this claim.

Proof. Inclusion $\supseteq$ in (15) always hold, meaning that every pair of behaviours unifiable in the right hand side of (15) is also unifiable in the left hand side. Thus it remains to show that, if the two conditions of Theorem 1 hold, then inclusion $\subseteq$ in (15) does too. Now, assume (17) and (18). Pick a pair (σ_1, σ_2) of behaviours which are unifiable in $P_1 \;_{(\rho}\|_{\rho)}\; P_2$. Then, by definition of $_{(\rho}\|_{\rho)}$, the pair $((\sigma_1)_\rho, (\sigma_2)_\rho)$ is unifiable in $(P_1)_\rho \,\|\, (P_2)_\rho$. Next, (18) guarantees that $(\sigma_1)_\rho \sqcup (\sigma_2)_\rho$ is a behaviour of $(P_1 \,\|\, P_2)_\rho$. Hence there must exist some pair (σ_1', σ_2') unifiable in $P_1 \;\|\; P_2$, such that $(\sigma_1' \sqcup \sigma_2')_\rho = (\sigma_1)_\rho \sqcup (\sigma_2)_\rho$. Using the same argument as before, we derive that $((\sigma_1')_\rho, (\sigma_2')_\rho)$ is also unifiable with respect to its associated (asynchronous) parallel composition, and $(\sigma_1')_\rho \sqcup (\sigma_2')_\rho =$

$(\sigma_1)_\rho \sqcup (\sigma_2)_\rho$. But $(\sigma'_1)_\rho$ is the restriction of $(\sigma'_1)_\rho \sqcup (\sigma'_2)_\rho$ to its events labeled by variables belonging to V_1, and similarly for $(\sigma'_2)_\rho$. Thus $(\sigma'_i)_\rho = (\sigma_i)_\rho$ for $i = 1, 2$ follows. Finally, using (17), we know that if (σ'_1, σ'_2) is such that, for $i = 1, 2$: $(\sigma'_i)_\rho = (\sigma_i)_\rho$, then: $\sigma'_i = \sigma_i$. Hence (σ_1, σ_2) is unifiable in $P_1 \parallel P_2$. $\diamond$

Corollary 1. *Let P_1 and P_2 be synchronous systems whose behaviors are equipped with some equivalence relation $\sim$, and assume that P_1 and P_2 are closed with respect to $\sim$. Then, the pair (P_1, P_2) satisfies condition (15) if it satisfies the following two conditions:*

$$\forall i \in \{1, 2\} : (P_i)_\rho \text{ is in bijection with } P_i/\sim \tag{19}$$

$$(P_1 \parallel P_2)_\rho = (P_1)_\rho \parallel (P_2)_\rho \tag{20}$$

where $P_i/\sim$ is the quotient of P_i modulo $\sim$.

Proof. Identical to the proof of Theorem 1 until the paragraph starting with "Finally". Finally, using (19), we know that if (σ'_1, σ'_2) is such that, for $i = 1, 2$: $(\sigma'_i)_\rho = (\sigma_i)_\rho$, then: $\sigma'_i \sim \sigma_i$. Hence (σ_1, σ_2) is unifiable in $P_1 \parallel P_2$, since all synchronous systems we consider are closed under $\sim$. $\diamond$

This result if of particular interest when $\sim$ is the equivalence modulo stuttering, defined by (4).

Running example, cont'd. Since P and Q possess a single behaviour, they clearly satisfy condition (17). However, the alternative condition (18) is violated: the left hand side is empty, while the right hand side is not. This explains why semantics is not preserved by desynchronization, for this example. In fact, it can be shown that the pair (P, Q) is not isochronous in the sense of [3,4]. More examples and counter-examples can be found in [5].

Discussion. In [10] the following result was proved. For P and Q two synchronous systems such that both P, Q, and $P \parallel Q$ are functional, clock-consistent, and with loop-free combinational part, then $P \parallel Q$ can be seen as a Kahn network—for our purpose, just interpret Kahn networks as functional asynchronous systems. This result applies to functional systems with inputs and outputs, it gives no help for partial designs or abstractions. Our conditions of endochrony and isochrony allows us to deal even with partial designs, not only with executable programs. Hence, they do represent effective techniques that can be used as part of the formal foundation for a successive-refinement design methodology.

As said before, this paper extends the ideas of [20] on desynchronization. A more naive "token-based" argument to explain GALS deployment is also found in [6], Sect. V.B. This argument is closely related to the use of Marked Graphs in [11] to justify GALS desynchronization in hardware.

Another example can be found in theory of latency-insensitive design [9]: here, if $P \parallel Q$ is a specification of a synchronous system and P and Q are *stallable* processes, then it is always possible to automatically derive two corresponding

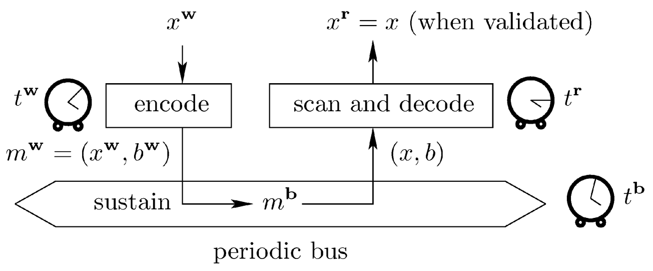

Fig. 1. The LTTA-protocol

patient processes P_p and Q_p that seamlessly compose to give a system implementation $P_p \parallel Q_p$ that preserves semantics while being also robust to arbitrary, but discrete, latency variations between P and Q. Again, $P_p \parallel Q_p$ is a correct deterministic executable system made of endochronous sub-systems. In fact, as the notion of stallable system and patient system correspond respectively to the notion of stuttering-invariant system and endochronous system, the extension to Theorem 1 subsumes the result presented in [9] on the compositionality of latency-insensitivity among patient processes.

4 Deploying Timed Synchronous Specifications over LTTA

Loosely Time-Triggered Architectures (LTTA) were introduced in [7] as a weakening of H. Kopetz' TTA. We revisit the results of [7] and complete them, by using the results from the previous section.

4.1 The LTTA-Protocol and Its Properties

See Figure 1 for an illustration of this protocol (the three watches shown indicate a different time, *they are not synchronized*). We consider three devices, the *writer,* the *bus,* and the *reader,* indicated by the superscripts $(.)^{\mathbf{w}}, (.)^{\mathbf{b}}$, and $(.)^{\mathbf{r}}$, respectively. Each device is activated by its own, approximately periodic, clock. The different clocks are *not* synchronized. In the following specification, the different sequences written, fetched, or read, are indexed by the set $\mathbf{N} = \{1, 2, 3, \ldots, n, \ldots\}$ of natural integers, and we reserve the index 0 for the initial conditions, whenever needed. Set $\mathbf{N}$ will serve to index the successive events of each individual signal, exactly as in our model of Section 2.1. Thus our informal description below is in agreement with our formal model. On the other hand, we believe that this informal description is quite natural.

The writer: At the time $t^{\mathbf{w}}(n)$ of the n-th tick of his clock, the writer generates a new value $x^{\mathbf{w}}(n)$ and a new alternating flag $b^{\mathbf{w}}(n)$ with:

$$b^{\mathbf{w}}(n) = \begin{cases} \textit{false} & \text{if } n = 0 \\ \textit{not } b^{\mathbf{w}}(n-1) & \text{otherwise} \end{cases}$$

and stores both in its private output buffer. Thus at any time t, the writer's output buffer content $m^{\mathbf{w}}$ is the last value that was written into it, that is the one with the largest index whose tick occurred before t:

$$m^{\mathbf{w}}(t) = (x^{\mathbf{w}}(n), b^{\mathbf{w}}(n)), \text{ where } n = \sup\{n' \mid t^{\mathbf{w}}(n') < t\} \tag{21}$$

The bus: At the time $t^{\mathbf{b}}(n)$ of its n-th clock tick, the bus fetches the value in the writer's output buffer and stores it, immediately after, in the reader's input buffer. Thus, at any time t, the reader's input buffer content offered by the bus, denote it by $m^{\mathbf{b}}$, is the last value that was written into it, i.e., the one written at the latest bus clock tick preceding t:

$$m^{\mathbf{b}}(t) = m^{\mathbf{w}}(t^{\mathbf{b}}(n)), \text{ where } n = \sup\{n' \mid t^{\mathbf{b}}(n') < t\} \tag{22}$$

The reader: At the time $t^{\mathbf{r}}(n)$ of its n-th clock tick, the reader copies the value of its input buffer into auxiliary variables $x(n)$ and $b(n)$:

$$(x(n), b(n)) = m^{\mathbf{b}}(t^{\mathbf{r}}(n))$$

Then, the reader extracts from the x sequence only the values corresponding to the indices for which b has alternated. This can be modeled thanks to the counter c, which counts the number of alternations that have taken place up to the current cycle. Hence, the value of the extracted sequence at index k is the value read at the earliest cycle when the counter c exceeded k:

$$\begin{aligned} c(n) &= \begin{cases} 0 & \text{if } n = 0 \\ c(n-1)+1 & \text{if } b(n) \neq b(n-1) \\ c(n-1) & \text{otherwise} \end{cases} \\ x^{\mathbf{r}}(k) &= x(n), \text{ where } k = c(n) \end{aligned} \tag{23}$$

Call LTTA-protocol the protocol defined by the formulas (21,22,23).

Theorem 2 ([7]). *Let the writing/bus/reading be systems with physically periodic clocks of respective periods $w/b/r$. Then, the* LTTA*-protocol satisfies the following property whatever the written input sequence is:*

$$\forall k : x^{\mathbf{r}}(k) = x^{\mathbf{w}}(k), \tag{24}$$

iff the following conditions hold:

$$w \geq b \text{ , and } \left\lfloor \frac{w}{b} \right\rfloor \geq \frac{r}{b} \text{ ,} \tag{25}$$

where, for x a real, $\lfloor x \rfloor$ denotes the largest integer $\leq x$.

Condition (24) means that the bus provides a coherent system of logical clocks. Note that, since $w \geq b$, then $w/2b < \lfloor w/b \rfloor$ follows. On the other hand, $\lfloor w/b \rfloor \leq w/b$, and $\lfloor w/b \rfloor \sim w/b$ for w/b large. Hence, for a fast bus, i.e. $b \sim 0$, the conditions (25) of Theorem 2 reduce to $w \gg b, w > r$. In [7], Theorem 2 is extended to clocks subject to time-varying jitter, assuming some bounds for this jitter, and the case of multiple-users is briefly discussed.

Let us now consider a more involved situation, where several units communicate, and where there are data dependencies between communicated values. A simple example would be that the reader, upon receiving $x(n)$ has to send back $y(n) = f(x(n))$ to the writer, by using the same protocol. Thus, the reader has also to maintain its own $b^{\mathbf{r}}$ and to update an output buffer $m^{\mathbf{r}}$. It now computes, at the time $t^{\mathbf{r}}(n)$ of its n-th clock tick:

$$\begin{aligned}
(x(n), b(n)) &= m^{\mathbf{b}}(t^{\mathbf{r}}(n)) \\
c(n) &= \begin{cases} 0 & \text{if } n = 0 \\ c(n-1)+1 & \text{if } b(n) \neq b(n-1) \\ c(n-1) & \text{otherwise} \end{cases} \\
x^{\mathbf{r}}(k) &= x(n) \\
y^{\mathbf{r}}(k) &= f(x^{\mathbf{r}}(k)) \\
b^{\mathbf{r}}(k) &= \begin{cases} \mathit{false} & \text{if } k = 0 \\ \mathit{not}\ b^{\mathbf{r}}(k-1) & \text{otherwise} \end{cases} \\
m^{\mathbf{r}}(t^{\mathbf{r}}(n)) &= (y^{\mathbf{r}}(k), b^{\mathbf{r}}(k))\,, \ \text{where } k = c(n)
\end{aligned}$$

This means that the computations indexed by k are only performed when the counter is increased, i.e., when the reader sees a bit alternation coming from the writer.

Timing issues: Theorem 2 says that, for the sake of protocol correctness, the reader should be faster than the writer. But now, the reader is also a writer and conversely. Hence, we need to distinguish between the period r_i at which any of these actors (reader or writer) i reads, and the period w_i at which the same actor i writes (updates its output buffer). This yields the following condition:

$$\forall i, j, \ \left\lfloor \frac{w_i}{b} \right\rfloor \ \geq \ \frac{r_j}{b}$$

Comparison with Lamport's logical clocks [18]: When several communicating actors are involved, the LTTA protocol is very reminiscent of Lamport's logical clock synchronization: in Lamport's protocol, each actor maintains a logical clock and when sending a message, time-stamps it with this clock. When receiving a message, the receiver compares its own clock with the time-stamp of the message. If its own clock is late with respect to the received time-stamps, it updates it. We could have stated the LTTA protocol similarly, by time-stamping the messages by the values of the counter c. Yet, the problem with Lamport's time-stamps is that they are ever increasing and, thus, subject to overflow. Here, the knowledge of periods and the properties of the LTTA architecture allow us to abstract these time-stamps into boolean ones:

$$b(n) = \begin{cases} \mathit{false} & \text{if } c(n) = 0 \text{ modulo } 2 \\ \mathit{true} & \text{otherwise} \end{cases}$$

4.2 Correct Ltta Deployment

Here, our work is slightly more involved. The reason is that the systems considered also involve physical time. And they involve physical time in two forms: absolute physical time and approximate physical time as provided by the imperfect watches. More precisely, in analyzing the deployment of synchronous designs over Ltta, the following notions need to be considered:

- Synchronous logical time to index the successive reactions of the synchronous specification.
- Physical global time as provided by an ideal and perfect clock. Corresponding dates can be used as part of the system specification.
- Actual local physical time as provided by each quasi-periodic writer/reader/bus clock (recall these are only loosely synchronized).

These are different "times", for combination and use at different stages of the design. As expected, heterogeneous systems will solve the problem.

The Ideal Design. Our reference is the so-called *ideal design.* It consists of a pair of timed synchronous specifications (P_1, P_2) for deployment over an ideal timed synchronous channel. The ideal channel is denoted by *Id*, it implements the *logically* instantaneous broadcast between P_1 and P_2, with some constant *physical* delay δ regarding dates. We adopt the following notational convention: if x_1 is output by P_1 toward P_2, the corresponding input for P_2 is denoted by x_2, and vice versa. Then, *Id* implements $x_2 := x_1$ with physical delay δ. Since we consider timed synchronous systems, we take as tag set: $\mathcal{T}_{\text{synch}} \times \mathcal{T}_{\varphi}$.

The Actual Deployment. It is modeled by $P_1 \ {}_{(\alpha}\| \ Ltta \ \|_{\alpha)} \ P_2$, where α is the canonical *date-preserving* desynchronization morphism:

$$\alpha \ : \ \mathcal{T}_{\text{synch}} \times \mathcal{T}_{\varphi} \ \longmapsto \ \{.\} \times \mathcal{T}_{\varphi},$$

and *Ltta*, the model of the Ltta medium, is a timed and asynchronous system with tag set $\{.\} \times \mathcal{T}_{\varphi}$. By Theorem 2, *Ltta* preserves the sequence of values of individual signals (this is the *flow invariance* condition of [20]). Regarding timing, if we assume bounded delay for the bus and bounded jitter due to asynchrony in sampling, then *Ltta* communication occurs with a nondeterministic delay within the bounds $[\delta - \varepsilon, \delta + \eta]$, where δ is the delay of the ideal medium. Semantics preserving, from ideal design to actual implementation, is captured as follows:

(a) The preservation of the functional semantics is modeled by the following request, which the reader should compare to relation (15):

$$P_1 \ {}_{((\alpha,t)}\|_{t)} \ Ltta \ {}_{(t}\|_{(t,\alpha))} \ P_2 \ \equiv \ P_1 \ {}_{(t}\|_{t)} \ Id \ {}_{(t}\|_{t)} \ P_2. \qquad (26)$$

In (26), t denotes the morphism consisting of the removal of physical time from the tag, α is our desynchronization morphism introduced before, and the pair (α, t) denotes the morphism consisting in jointly removing physical time and desynchronizing. Thus the right hand side of (26) is the reference untimed semantics of our design, whereas the left hand side is the actual untimed deployment semantics. Thus (26) is indeed the requirement of preserving the functional semantics.

(b) Regarding timing aspects, we consider separately the bounds $[\delta - \varepsilon, \delta + \eta]$, for the timing behaviour of *Ltta* for transmitting each individual message, asynchronously.

The following theorem holds. It refines Theorem 1 for the case of mixed synchronous/timed and asynchronous/timed systems. It relies on the property

$$Ltta_t = Id_{(\alpha,t)},$$

which is a reformulation of property (24) in Theorem 2. Its proof is omitted because it is a mild variation of the proof of Theorem 1.

Theorem 3. *The pair (P_1, P_2) satisfies condition (26) if it satisfies the following two conditions:*

$$\forall i \in \{1,2\} : (P_i)_{(\alpha,t)} \text{ is in bijection with } (P_i)_t \tag{27}$$

$$\left(P_1 \ {}_{(t}\|_{t)} \ Id \ {}_{(t}\|_{t)} \ P_2\right)_\alpha \equiv (P_1)_{(\alpha,t)} \ \| \ Id_{(\alpha,t)} \ \| \ (P_2)_{(\alpha,t)} \tag{28}$$

Discussion. Theorem 3 gives sufficient conditions for semantics preserving, that are *independent* from the precise form of *Ltta*, because only the ideal channel *Id* is involved. The conditions that guarantee semantics preserving are (almost) identical to those of Th. 1, thus endo/isochrony apply as well to this case.

5 Concluding Remarks

In this paper, we proposed a novel mathematical framework (an extension to the LSV tagged signal model) for handling heterogeneous reactive systems. The interest of this theory rests on the theorems it can provide. These theorems support effective techniques to generate automatically correct-by-construction adaptors, between two designs supported by different coordination paradigms.

References

1. R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas and O. Sokolsky. Hierarchical Modeling and Analysis of Embedded Systems. *Proc. of the IEEE,* 91(1), 11–28, Jan. 2003.
2. A. Benveniste. Some synchronization issues when designing embedded systems from components. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT'01,* T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 32–49, 2001.
3. A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99, Concurrency Theory, 10th Intl. Conference*, LNCS 1664, pages 162–177. Springer, Aug. 1999.

4. A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in dataflow synchronous languages: specification & distributed code generation. *Information and Computation,* 163, 125–171 (2000).
5. A. Benveniste, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli. Heterogeneous reactive systems modeling and correct-by-construction deployment. Technical Report UCB/ERL M03/23, Electronics Research Lab, University of California, Berkeley, CA 94720, June 2003.
6. A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The Synchronous Language Twelve Years Later. *Proc. of the IEEE*, 91(1):64–83, January 2003.
7. A. Benveniste, P. Caspi, P. Le Guernic, H. Marchand, J-P. Talpin and S. Tripakis. A Protocol for Loosely Time-Triggered Architectures. In *Embedded Software. Proc. of the 2nd Intl. Workshop, EMSOFT 2002*, Grenoble, France, Oct. 2002.
8. G. Berry. The Foundations of Esterel. MIT Press, 2000.
9. L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of Latency-Insensitive Design. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, September 2001.
10. P. Caspi, "Clocks in Dataflow languages", *Theoretical Computer Science,* vol. 94:125–140, 1992.
11. J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. A concurrent model for de-synchronization. In *Proc. Intl. Workshop on Logic Synthesis*, May 2003.
12. L. de Alfaro and T.A. Henzinger. Interface Theories for Component-Based Design. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT'01,* T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 32–49, Springer Verlag, 2001.
13. J. Eker, J.W. Janneck, E.A. Lee, J. Liu, J. Ludwig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity–The Ptolemy approach. *Proc. of the IEEE,* 91(1), 127–144, Jan. 2003.
14. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Data Flow Programming Language LUSTRE. *Proc. of the IEEE*, 79(9):1305–1320, Sep. 1991.
15. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
16. G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. *Proc. of the IEEE,* 91(1), 127–144, Jan. 2003.
17. H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers. 1997. ISBN 0-7923-9894-7.
18. L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communication of the ACM*, 21:558–565, 1978.
19. P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real-time applications with SIGNAL. *Proc. of the IEEE*, 79(9):1326–1333, Sep. 1991.
20. P. Le Guernic, J.-P. Talpin, J.-C. Le Lann, Polychrony for system design. *Journal for Circuits, Systems and Computers.* World Scientific, April 2003.
21. E.A. Lee and A. Sangiovanni-Vincentelli. A Framework for Comparing Models of Computation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* 17(12), 1217–1229, Dec. 1998.
22. E.A. Lee and Y. Xiong. System-Level Types for Component-Based Design. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT'01,* T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 32–49, Springer Verlag, 2001.
23. M. Mokhtari and M. Marie. Engineering Applications of MATLAB 5.3 and SIMULINK 3. Springer Verlag, 2000.