2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

A Simulation Condition for Correct Asynchronous Implementation of Synchronous Design

S. Xu, R. Kumar, S. Jiang, and S. Ramesh

Abstract— **We study the problem of "desynchronization", i.e., semantics-preserving "asynchronous implementation" of a "synchronous design". In a synchronous design, system components (which we model as input-output automata (I/O-automata)) communicate over synchronous channels and their combined behavior can be described using synchronous composition, whereas in an asynchronous implementation, communication among components occurs over asynchronous channels (which we also model as I/O-automata) and the behavior of an asynchronous implementation can be described using asynchronous composition. The presence of asynchronous communication can result in additional behavior that is not present under synchronous communication and can thus cause the semantics of a synchronous design to be not preserved under asynchronous implementation. We formalize the notion of system response to an input sequence and by using it, define a criterion for correct desynchronization. We define the simulation of I/O-automata, and argue that the simulation of the asynchronous implementation by a synchronous design is sufficient to guarantee the correctness of desynchronization. This is a new way of characterizing the correctness of desynchronization (as compared to the "iso-/endo-chrony" type conditions proposed in previous works). Under the practical assumption that the communication delay is bounded, the proposed simulation condition is algorithmically verifiable.**

Keywords: Synchronous Language, Asynchronous Implementation, GALS, Desynchronization, Discrete-Event Systems, Input-Output Automata

I. Introduction

Synchronous programming languages, such as Esterel, Lustre and Signal, have been proposed as means for the design of real-time systems which react to the environment (see for example [1], [2], [3]). These synchronous languages rely on synchronous hypothesis that computation and communication are "zero-time", and a program can react to the external inputs instantaneously [4]. Synchronous languages offer numerous advantages in the real-world designs, simplifying specification, synthesis and verification, and providing the designers with ideal primitives [5]. However, the real-life architectures do not obey the ideal model of perfect synchrony, namely, "zero-time" computation and instantaneous broadcast communication [6]. Many systems such as industrial control systems are physically distributed and hence asynchronous [7]. In the hardware world, when the circuit size becomes large, maintaining global synchrony becomes quite expensive or infeasible [8]. Thus the synchronous hypothesis can not usually be satisfied, which leads to a disconnect between a synchronous design and its physical (non-synchronous) implementation.

The research was supported in part by the National Science Foundation under the grants NSF-ECS-0424048, NSF-ECS-0601570 and NSF-ECCS-08013763.

S. Xu and R. Kumar are with the Department Electrical and Computer Engineering, Iowa State University, Ames, IA; email: syxu,rkumar@iastate.edu

S. Jiang and S. Ramesh are with General Motors R&D, Warren, MI, and India respectively; email: shengbing.jiang,ramesh.s@gm.com

A. Desynchronization Problem

In a physical implementation, since the presence of asynchronous communication can result in additional behavior that is not present under synchronous design, the semantics of a synchronous design may be not preserved under the implementation. Thus a primary issue of desynchronization is the correctness of the implementation, i.e, the preservation of the synchronous semantics [9]. The efficiency of an implementation is also an important issue, namely, the amount of overhead required for guaranteeing the correctness [10]. The following example [11], [12] shows the loss of semantics while performing desynchronization.

Example 1: Suppose a system, as shown in Figure 1, consists of an arbiter, an emitter and two counters (1bit and 2bit in restart mode and resume mode respectively) operating concurrently.

Each time the message p arrives from the environment at the arbiter, it starts or stops the counters by sending the messages st_2, sp_2, st_1, sp_1 alternately, where st_i (resp., sp_i) starts (resp., stops) counter-i. Each time the arbiter receives a message a from the environment, it transmits that to a counter that is currently running (and if no counter is currently running the message a is ignored). A running counter counts the occurrences of message a relayed by the arbiter and transmits the count value as out to the emitter when it reaches the maximum count value ("2" for 1bit counter and "4" for 2bit counter). The emitter then emits that value to the environment. Note the 1bit counter is reset when the stop signal sp_1 arrives, but no resetting occurs for the 2bit counter upon the arrival of the stop signal sp_2.

Suppose the message sequence received from the environment is "$paaaappaappaap$". In the synchronous setting, the sequence of outputs will be "42". A possible execution sequence resulting from the same message sequence in an asynchronous setting is depicted in a message sequence chart format in Figure 2. In the asynchronous scenario, a message a that departed earlier than a message sp_2 ends up arriving later than the message sp_2. This causes the sequence of outputs to be "24". Clearly, the desynchronization leads to a loss of semantics of the synchronous setting.

The main reason for the loss of semantics is that not all variables are "present" in all communications,

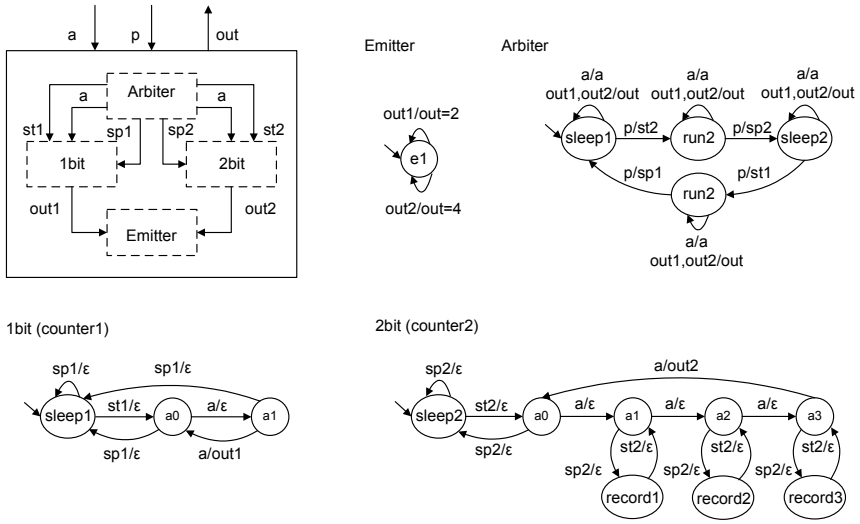

Fig. 1. Arbiter-Emitter-Counters system and their models

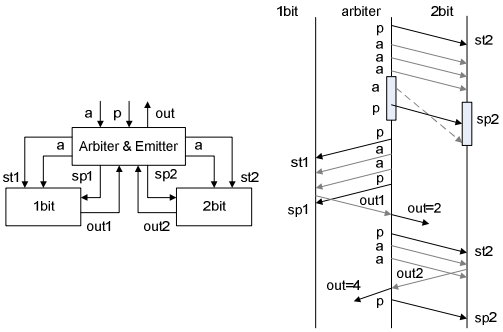

Fig. 2. Message sequence chart of a possible execution

and the absence of certain variables cannot be sensed in the asynchronous setting [9]. For example, in a communication from the arbiter to the counter-i, only one of st_i or sp_i or a is present at any instant, while the others are absent. For illustration, consider the prefix "$paaaap$" of the message sequence mentioned earlier. Then the following sequence is received by the channel from the arbiter: $(\epsilon, st_2, \epsilon)(a, \epsilon, \epsilon)(a, \epsilon, \epsilon)(a, \epsilon, \epsilon)(a, \epsilon, \epsilon)(\epsilon, \epsilon, sp_2)$, where ϵ denotes "absent".

Assuming the first four messages are delivered to the counter-2 in the same order as they were received by the channel, the channel state at this point is (a, ϵ, sp_2). Due to the asynchrony of communication, at this point, it is possible that sp_2 is delivered by the channel to counter-2 before the delivery of a, resulting in the channel state of (a, ϵ, ϵ). (Another possibility is "in-order" delivery of a as in the synchronous setting resulting in the channel state of $(\epsilon, \epsilon, sp_2)$.) This scenario is not possible in the synchronous setting and leads to the loss of correctness under desynchronization, i.e., an output 2 proceeding an output 4.

B. Related Work

Prior works on the problem of desynchronization include the latency-insensitive systems [13], the notions of iso-/endo-chrony [6], and the notions of weak iso-/endo-chrony [10].

A *latency-insensitive* system is a synchronous distributed system composed of functional components that exchange data on communication channels using a latency insensitive protocol which guarantees the preservation of the synchronous semantics [13]. Such a protocol introduces delays and reduces the speed of the system. Benveniste el al. [6] proposed the concepts of isochrony and endochrony, where *isochrony* requires that when a pair of transitions are not synchronizable, i.e., they disagree on the value of a shared variable, then the shared nonempty variable should carry a contradictory value. Whereas, *endochrony* requires that a system should be able to infer presence/absence of all its input variables incrementally as a function of the current state and the values of the input variables currently available. This property is used for "resynchronization", i.e., reconstruction of a synchronous behavior from the asynchronous observation of the system behavior. In [14], it is shown that using wrappers a system (resp., pair of systems) can always be made endochronous (resp., isochronous). A drawback of endochrony is that it is not compositional, i.e., composition of endochronous system need not be endochronous. In [10], Potop-Butucaru et al. proposed the weaker notions of weak endochrony (which is compositional) and weak isochrony, which together guarantee the correctness of desynchronization. Further as the case with iso-/endo-chrony, wrappers can be designed to make systems weakly iso-/endo-chronous.

C. Contribution

In this paper, we study the problem of correct desynchronization over a "GALS" (globally asynchronous and locally synchronous) architecture, in which the synchronous components interact by exchanging signals connected by asynchronous channels. We model individual *systems* as well as asynchronous *channels* using input-output automata (I/O automata), and use their compositions to come up with the models of the synchronous design as well as

the asynchronous implementation. We define the notion of simulation of I/O-automata and show that the simulation of the asynchronous implementation by the synchronous design suffices for the correctness of desynchronization. This condition is different from the prior proposed condition of isochrony [6], and can be verified under the practical assumption that the communication delay is bounded.

II. Notation and Preliminaries

In this section, we define several notions which are needed to formulate the problem of desynchronization and for proposing a solution.

A. System Model

In GALS architecture, the systems communicate with each other and their environments via communication channels. We model each system and each communication channel as an input/output automaton (I/O-automaton). In the following, the notion $\overline{A}$ is used to denote $A \cup \{\vec{\epsilon}\}$, for any vector alphabet A. Note $\vec{\epsilon}$ serves as the identity of concatenation (i.e., $\alpha\vec{\epsilon} = \vec{\epsilon}\alpha = \alpha$ for any $\alpha \in \overline{A}^*$) and $\overline{A}^* = A^*$.

Definition 1: An input/output automaton (I/O-automaton) is a six-tuple $G = (X, U, Y, T, X_0, X_m)$, where

- X is the set of state-vectors (can be r-dimensional)
- U is the set of input-vectors (can be p-dimensional)
- Y is the set of output-vectors (can be q-dimensional)
- $T \subseteq X \times \overline{U} \times \overline{Y} \times X$ is the transition set, i.e., each $t \in T$ is a four-tuple, $t = (o_t, \vec{u}_t, \vec{y}_t, d_t)$, where
  - $o_t \in X$ is the origin state of transition t,
  - $\vec{u}_t \in \overline{U}$ is the input for transition t,
  - $\vec{y}_t \in \overline{Y}$ is the output from transition t,
  - $d_t \in X$ is the destination state of transition t
- $X_0 \subseteq X$ is the set of initial state-vectors
- $X_m \subseteq X$ is the set of marked (or accepting) states.

For any $\vec{u} \in \overline{U}$ and $\vec{y} \in \overline{Y}$, some components of $\vec{u}$ and $\vec{y}$ can be "absent", i.e., letting $\vec{u}(i)$ and $\vec{y}(j)$ denote the ith and jth components of $\vec{u}$ and $\vec{y}$ respectively, we can have $\vec{u}(i) = \epsilon$ or $\vec{y}(j) = \epsilon$.

An I/O-automaton $G = (X, U, Y, T, X_0, X_m)$ is said to be deterministic if for all $x \in X, \vec{u} \in U$,

- $|X_0| = 1$
- $|\{t \in T | o_t = x, \vec{u}_t = \vec{u}\}| \leq 1$
- $|\{t \in T | o_t = x, \vec{u}_t = \vec{\epsilon}\}| = 0$.

In the following, a concatenation of two "vector-sequences" is taken to be the component-wise concatenation. For example, for $\vec{u}_1, \vec{u}_2 \in U^*$, $\vec{u}_1\vec{u}_2(i) = \vec{u}_1(i)\vec{u}_2(i)$, where $i \in \{1, \cdots, p\}$.

Definition 2: The input-output behavior of a system G is described by its generated input-output language, $L(G) \subseteq (\overline{U} \times \overline{Y})^*$, where

$$(\vec{u}_0, \vec{y}_0)(\vec{u}_1, \vec{y}_1) \ldots (\vec{u}_n, \vec{y}_n) \in L(G),$$

if and only if for each $i \leq n$ ($n \in \mathcal{N}$, the set of natural numbers), there exists transition $t_i = (o_i, \vec{u}_i, \vec{y}_i, d_i)$ such that

- $o_0 \in X_0$, and
- $d_i = o_{i+1}$ for $i < n$.

Further if $d_n \in X_m$, then $(\vec{u}_0, \vec{y}_0)(\vec{u}_1, \vec{y}_1) \ldots (\vec{u}_n, \vec{y}_n) \in L_m(G)$, which is the marked input-output language of G.

The response of G to an input sequence is defined as follows.

Definition 3: Given G, the set of all generated (resp. accepted) responses to an input sequence, $\mu \in U^*$, denoted $L(G, \mu) \subseteq Y^*$ (resp. $L_m(G, \mu)$), is defined as:

$$\begin{aligned} L(G,\mu) &:= \{\gamma \in Y^* \mid \exists (\vec{u}_0, \vec{y}_0) \ldots (\vec{u}_m, \vec{y}_m) \in L(G), \\ &\qquad \text{and } \mu = \vec{u}_0 \ldots \vec{u}_m, \gamma = \vec{y}_0 \ldots \vec{y}_m\} \\ L_m(G,\mu) &:= \{\gamma \in Y^* \mid \exists (\vec{u}_0, \vec{y}_0) \ldots (\vec{u}_m, \vec{y}_m) \in L_m(G), \\ &\qquad \text{and } \mu = \vec{u}_0 \ldots \vec{u}_m, \gamma = \vec{y}_0 \ldots \vec{y}_m\} \end{aligned}$$

Note in the above definition, we have used the fact that for any trace $\alpha \in U^* \cup Y^*$, $\alpha\vec{\epsilon} = \vec{\epsilon}\alpha = \alpha$. In the following, we only consider the system components which are modeled as deterministic I/O-automata.

B. Channel Model

Without loss of generality, it is assumed that each channel carries a *single* signal. If a channel represents a "bus", the entire packet communicated over the bus is viewed as a single signal. Since multiple signals may be shared between two systems, we view two systems to be connected by a "vector-channel". While each individual channel of a vector-channel has a FIFO behavior, their asynchronous nature (i.e., variable delay) causes a vector-channel to unnecessarily behave in a FIFO manner. Next, an asynchronous vector-channel is presented by I/O-automaton. Unless otherwise stated, we will use "channel" to refer to an "asynchronous vector-channel".

Definition 4: An asynchronous vector-channel (or simply a channel) with inputs/outputs belonging to a set $Z = \prod_i Z_i$ is an I/O-automaton,

$$C^a = (\prod_i Z_i^*, Z, Z, T, \{\vec{\epsilon}\}, \{\vec{\epsilon}\})$$

possessing transitions of the following form:

$$(\zeta, \vec{z}, \vec{\epsilon}, \zeta\vec{z}); \quad \text{and} \quad (\zeta, \vec{\epsilon}, \vec{z}, \zeta\backslash\vec{z}).$$

Here the channel state is a sequence $\zeta \in \prod_i Z_i^*$, denoting the sequence that has arrived in the channel but has not departed yet, i.e., the sequence buffered in the channel. On an arrival transition $(\zeta, \vec{z}, \vec{\epsilon}, \zeta\vec{z})$, the channel state ζ is appended with a new input $\vec{z} \in Z$ changing it to $\zeta\vec{z}$, whereas on a departure transition $(\zeta, \vec{\epsilon}, \vec{z}, \zeta\backslash\vec{z})$, an output $\vec{z} \in Z$ is removed from the head of the channel state ζ changing it to $\zeta\backslash\vec{z}$. Here the notation "$\backslash$" is the component-wise "after" operation, i.e., $\zeta\backslash\vec{z}$ is the "suffix vector" whose ith component is obtained by removing the prefix $\vec{z}(i)$ from $\zeta(i)$. Note that in an arrival (resp., departure) transition nothing is outputted (resp., inputted). Thus arrivals and departures occur asynchronously.

Remark 1: Note that in an asynchronous channel, multiple outputs may be possible for one input due to a variable delay among the individual channels of a vector-channel, i.e., there may exist multiple transitions at some state ζ, each with the same input $\vec{z}_1$ but with different outputs $\vec{z}_2$. This causes the asynchronous channel model to be nondeterministic in general (as can be seen in Figure 1).

Remark 2: The buffering length of an asynchronous channel is bounded in practice. Supposing the bound is n, the number of possible channel states are $\prod_i(|Z_i|+1)^n$. Thus asynchronous channels can be represented by finite-state I/O-automata.

Remark 3: Note in the synchronous mode of operation the departure of a channel output occurs before the arrival of the next channel input. Thus a synchronous channel can be modeled as a special type of asynchronous channel with I/O-automaton model $C^s = (\overline{Z}, Z, Z, \{(\vec{\epsilon},\vec{z},\vec{\epsilon},\vec{z}),(\vec{z},\vec{\epsilon},\vec{z},\vec{\epsilon})|z \in Z\}, \{\vec{\epsilon}\}, \{\vec{\epsilon}\})$, where any arrival transition of the form $(\vec{\epsilon},\vec{z},\vec{\epsilon},\vec{z})$ is immediately followed by the corresponding departure transition $(\vec{z},\vec{\epsilon},\vec{z},\vec{\epsilon})$.

C. Synchronous Vs. Asynchronous Composition

When two systems $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i}), i = 1,2$, interact with each other by the way of sharing some inputs and outputs, their input and output sets are further partitioned as follows ($i, j = 1,2, i \neq j$):

$$U_i = P_i \times S \times I_{ji}, Y_i = E_i \times I_{ij},$$

where

- P_i is private external inputs to system i
- S is shared external inputs between systems i and j
- E_i is external outputs of system i
- I_{ij} is internal outputs of system i to system j.

When G_1 and G_2 interact synchronously, the "communication variables" I_{ij} travel over a synchronous channel C^s_{ij}. In the synchronous mode of operation, the combined system is given by the synchronous composition of G_1 and G_2:

Definition 5: Given I/O-automata, $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i}), i = 1,2$, their synchronous composition is given by, $G_1||G_2 := (X_1 \times X_2, P_1 \times P_2 \times S, E_1 \times E_2, T_{||}, X_{0,1} \times X_{0,2}, X_{m,1} \times X_{m,2})$, where $T_{\|} = T^a \cup T^b \cup T^c$, satisfying:

$$\begin{aligned}
&((o_1,o_2),(\vec{p}_1,\vec{p}_2,\vec{s}),(\vec{e}_1,\vec{e}_2),(d_1,d_2)) \in T^a \\
\Leftrightarrow\ &\exists(\vec{i}_{12},\vec{i}_{21}) \in \overline{I}_{12} \times \overline{I}_{21} - \{(\vec{\epsilon},\vec{\epsilon})\} : \\
&(o_1,(\vec{p}_1,\vec{s},\vec{i}_{21}),(\vec{e}_1,\vec{i}_{12}),d_1) \in T_1, \\
&(o_2,(\vec{p}_2,\vec{s},\vec{i}_{12}),(\vec{e}_2,\vec{i}_{21}),d_2) \in T_2.
\end{aligned}$$

$$\begin{aligned}
&((o_1,o_2),(\vec{p}_1,\vec{\epsilon},\vec{s}),(\vec{e}_1,\vec{\epsilon}),(d_1,o_2)) \in T^b \\
\Leftrightarrow\ &(o_1,(\vec{p}_1,\vec{s},\vec{\epsilon}),(\vec{e}_1,\vec{\epsilon}),d_1) \in T_1.
\end{aligned}$$

$$\begin{aligned}
&((o_1,o_2),(\vec{\epsilon},\vec{p}_2,\vec{s}),(\vec{\epsilon},\vec{e}_2),(o_1,d_2)) \in T^c \\
\Leftrightarrow\ &(o_2,(\vec{p}_2,\vec{s},\vec{\epsilon}),(\vec{e}_2,\vec{\epsilon}),d_2) \in T_2.
\end{aligned}$$

Remark 4: It is obvious that synchronous composition of two I/O-automata is also an I/O-automaton, and further it can be verified that the synchronous composition operation is commutative as well as associative.

Note due to the non-buffering nature of the synchronous channel, $G_1\|G_2$ is isomorphic to $G_1\|C^s_{12}\|G_2\|C^s_{21}$, and so there is no need to include C^s_{ij} explicitly in the definition of synchronous composition of G_1 and G_2.

When G_1 and G_2 interact asynchronously the "communication variables" I_{ij} travel over an asynchronous channel C_{ij}. In such a mode, the combined system is given by the synchronous composition of G_1, G_2 and the asynchronous channels:

Definition 6: Given I/O-automata, $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i}), i = 1,2$, their asynchronous composition is given by,

$$G_1||^a G_2 := G_1||C^a_{12}||G_2||C^a_{21},$$

where $C^a_{ij}, i, j = 1,2, i \neq j$ is an I/O-automaton representing the asynchronous channel between systems i and j, i.e.,

$$C^a_{ij} = (\prod_k I_{ij}(k)^*, I_{ij}, I_{ij}, T_{ij}, \{\vec{\epsilon}\}, \{\vec{\epsilon}\}).$$

Remark 5: Note that the asynchronous composition can be obtained by first encapsulating the output channels within a system to obtain an "asynchronized" system, and next taking the synchronous composition of the asynchronized systems. Formally, in order to define the asynchronous composition of $\{G_i, i \leq n\}$, we define for each i, the asynchronized system, $G^a_i := G_i\|_{j\neq i}C^a_{ij}$, which is the encapsulation of G_i and all its output channels. Then $\|^a_i G_i = \|_i G^a_i$. In this manner, the asynchronous composition is a type of synchronous composition (of systems encapsulating their output channels), and so enjoys the same set of properties such as commutativity and associativity.

Due to the nondeterminism of the asynchronous channels, the asynchronous composition of G_1 and G_2 is in general nondeterministic.

D. Notion of (Bi)-Simulation

The notion of (bi)-similarity of two systems was introduced by Milner [15]. Here we apply the definition to the context of I/O-automata [16].

Definition 7: Given two I/O-automata $G_i = (X_i, U, Y, T_i, X_{0,i}, X_{m,i})$, $i = 1,2$, a binary relation $\Phi \subseteq (X_1 \cup X_2) \times (X_1 \cup X_2)$ is a simulation relation if $(x_1,x_2) \in \Phi$ implies
$x_1 \in X_{m,1} \Rightarrow x_2 \in X_{m,2}$ and
$\forall t_1 = (x_1, \vec{u}_1, \vec{y}_1, \overline{x}_1) \in T_1 \cup T_2$,
$\exists t_{21}t_{22}\ldots t_{2k}$:
$\{t_{2j} = (x_{2j}, \vec{u}_{2j}, \vec{y}_{2j}, x_{2j+1}), j \leq k\} \subseteq T_1 \cup T_2$,
$\exists j \leq j' \leq k : [\vec{u}_{2j} = u_1, \vec{y}_{2j'} = y_1] \wedge \forall \bar{j} \neq j : [\vec{u}_{2\bar{j}} = \vec{\epsilon}\,], \forall \bar{j} \neq j' : [\vec{y}_{2\bar{j}} = \vec{\epsilon}\,]$,
$x_{21} = x_2, (\overline{x}_1, x_{2k+1}) \in \Phi$.

Note when $x_i \in X_i \subseteq X_1 \cup X_2$, $(x_i, \vec{u}, \vec{y}, x'_i) \in T_1 \cup T_2$ if and only if $(x_i, \vec{u}, \vec{y}, x'_i) \in T_i$. x_1 is said to be *simulated* by x_2, denoted $x_1 \sqsubseteq_\Phi x_2$, if exists a simulation relation Φ such that $(x_1,x_2) \in \Phi$. x_1 and x_2 are said to be *similar* (or simulation equivalent), denoted $x_1 \sim_\Phi x_2$, if $x_1 \sqsubseteq_\Phi x_2$ and $x_2 \sqsubseteq_\Phi x_1$. When Φ is *symmetric*, the similar x_1 and x_2 are called *bisimilar*, denoted $x_1 \simeq_\Phi x_2$.

X_1 is said to be simulated by X_2, denoted $X_1 \sqsubseteq_\Phi X_2$, if exists a simulation relation Φ such that for each $x_1 \in X_1$, exists $x_2 \in X_2$ and $(x_1,x_2) \in \Phi$. We write $X_1 \sim_\Phi X_2$ if $X_1 \sqsubseteq_\Phi X_2$ and $X_2 \sqsubseteq_\Phi X_1$. For two I/O automata $G_i = (X_i, U, Y, T_i, X_{0,i}, X_{m,i})$, $i = 1,2$, G_1 is said to

be simulated by (resp., similar or bisimilar to) G_2, denoted $G_1 \sqsubseteq_\Phi G_2$ (resp., $G_1 \sim_\Phi G_2$ or $G_1 \simeq_\Phi G_2$), if exists a simulation (resp., similarity or bisimulation) relation Φ such that $X_{0,1} \sqsubseteq_\Phi X_{0,2}$ (resp., $X_{0,1} \sim_\Phi X_{0,2}$ or $X_{0,1} \simeq_\Phi X_{0,2}$). Sometimes we omit the subscript Φ meaning that it is understood to exist.

Remark 6: It follows from the definition of the simulation relation Φ that if $(x_1, x_2) \in \Phi$, then $x_1 \in X_{m,1}$ implies $x_2 \in X_{m,2}$ and

$$\begin{array}{l}\forall t_{11}t_{12}\dots t_{1k_1}: \\ \{t_{1j} = (x_{1j}, \vec{u}_{1j}, \vec{y}_{1j}, x_{1j+1}), j \le k_1\} \subseteq T_1 \cup T_2, \\ x_{11} = x_1, \vec{u}_{11}\dots\vec{u}_{1k_1} \in U \\ \exists t_{21}t_{22}\dots t_{2k_2}: \\ \{t_{2j} = (x_{2j}, \vec{u}_{2j}, \vec{y}_{2j}, x_{2j+1}), j \le k_2\} \subseteq T_1 \cup T_2, \\ x_{21} = x_2, \vec{u}_{11}\dots\vec{u}_{1k_1} = \vec{u}_{21}\dots\vec{u}_{2k_2}, \\ \vec{y}_{11}\dots\vec{y}_{1k_1} = \vec{y}_{21}\dots\vec{y}_{2k_2}, (x_{1k_1}, x_{2k_2}) \in \Phi.\end{array}$$

III. Condition for Correct Desynchronization

In this section, we present a condition that guarantees that the semantics of a synchronous design is preserved under its asynchronous implementation. We first present the criterion for a correct desynchronization.

Definition 8: Given I/O automata $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i})$, $i \le n$, their asynchronous implementation is *correct* with respect to their synchronous design if

$$\forall \mu \in U^* : L_m(\|_i^a G_i, \mu) = L_m(\|_i G_i, \mu),$$

where U is the set of inputs of the combined system.

The above definition of correct desynchronization requires that for each input sequence, the set of accepted responses of an asynchronous implementation is the same as that of the synchronous design. Note that the equality of *accepted* responses is required as this ensures that there is nothing pending to be delivered in the communication channels.

Remark 7: It should be noted that a tighter criterion for correctness may be specified in a timed-setting by requiring that the response additionally meets some timing constraints, such as an output appears within a certain delay of the corresponding input. Adding such timing constraints, however, does not change the conceptual nature of the problem.

Using the simulation relation of I/O-automata defined earlier, we propose a sufficient condition for correct desynchronization. We first note that the synchronous composition of systems is always simulated by their asynchronous composition. This requires establishing a few properties first. Lemma 1 states that the synchronous composition preserves the simulation relation. (A similar result can be found in [17] but our definition of synchronous composition is different from that in [17] and so the result in [17] does not automatically imply the result of Lemma 1.) Due to space considerations, the proof is omitted.

Lemma 1: Given I/O-automata $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i})$, and $G_i' = (X_i', U_i, Y_i, T_i', X_{0,i}', X_{m,i}')$, $i = 1, 2$, if there exist simulation relations Φ_i such that $G_i \sqsubseteq_{\Phi_i} G_i'$, then $G_1 \| G_2 \sqsubseteq G_1' \| G_2'$.

The following corollary follows from Lemma 1 and the fact that synchronous composition is associative.

Corollary 1: Given I/O automata $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i})$ and $G_i' = (X_i', U_i, Y_i, T_i', X_{0,i}', X_{m,i}')$, $i \le n$, if $G_i \sqsubseteq_{\Phi_i} G_i'$ for each $i \le n$, then $\|_i G_i \sqsubseteq \|_i G_i'$.

Using the corollary above, next we prove that the synchronous composition of systems is simulated by their asynchronous composition.

Theorem 1: Given I/O-automata $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i})$, $i \le n$, it holds that $\|_i G_i \sqsubseteq \|_i^a G_i$.

Proof: Recall that $\|_i^a G_i$ can be written as $\|G_i^a$, where $G_i^a = G_i\|_{j \ne i} C_{ij}^a$, and C_{ij}^a is the asynchronous channel from G_i to G_j. In view of Corollary 1, it suffices to show that $G_i \sqsubseteq G_i^a$ for each i.

Recall also that G_i is isomorphic to $G_i\|_{j \ne i} C_{ij}^s$, where C_{ij}^s is the synchronous channel from G_i to G_j. Further each C_{ij}^s has its initial state marked, and transitions of the form $(\vec{\epsilon}, \vec{z}, \vec{\epsilon}, \vec{z})$ followed by $(\vec{z}, \vec{\epsilon}, \vec{z}, \vec{\epsilon})$, which are also present as transitions at the marked initial state of C_{ij}^a. So it follows that $C_{ij}^s \sqsubseteq C_{ij}^a$. Applying Corollary 1, we can conclude that $G_i \equiv G_i\|_{j \ne i} C_{ij}^s \sqsubseteq G_i\|_{j \ne i} C_{ij}^a = G_i^a$, where the notation $\equiv$ denotes the isomorphism. Another application of Corollary 1 provides, $\|_i G_i \sqsubseteq \|_i G_i^a = \|_i^a G_i$. ∎

For presenting the main result of the paper, we need the result of Lemma 2 presented below. Lemma 2 states that the existence of simulation relation implies system-response containment. (A result similar as that of Lemma 2 is given in [17]. However the two settings are not the same, and so the result of Lemma 2 does not automatically follow from that in [17].) Due to space considerations, the proof is omitted.

Lemma 2: Given I/O-automata $G = (X, U, Y, T, X_0, X_m)$ and $G' = (X', U, Y, T', X_0', X_m')$, if $G \sqsubseteq_\Phi G'$, then $\forall \mu \in U^*$, $L(G, \mu) \subseteq L(G', \mu)$ and $L_m(G, \mu) \subseteq L_m(G', \mu)$.

Now we are ready to state the main result of this paper that when the asynchronous composition of systems is simulated by their synchronous composition, the desynchronization will be correct.

Theorem 2: Given I/O-automata $G_i = (X_i, U_i, Y_i, T_i, X_{0,i}, X_{m,i})$, $i \le n$, $(n \in \mathcal{N})$, let $G^a := \|_i^a G_i$, $G^s := \|_i G_i$. If $G^a \sqsubseteq G^s$, then $\forall \mu \in U^*$, $L_m(G^a, \mu) = L_m(G^s, \mu)$, that is, the asynchronous implementation G^a is correct with respect to the synchronous specification G^s.

Proof: Since $G^a \sqsubseteq G^s$, from Lemma 2 we have that $\forall \mu \in U^*$, $L_m(G^a, \mu) \subseteq L_m(G^s, \mu)$. Also by Theorem 1, $G^s \sqsubseteq G^a$. It follows from a second application of Lemma 2 that $\forall \mu \in U^*$, $L_m(G^s, \mu) \subseteq L_m(G^a, \mu)$. Thus we obtain, $\forall \mu \in U^*$, $L_m(G^s, \mu) = L_m(G^a, \mu)$. ∎

Example 2: Figure 3 shows the synchronous and asynchronous compositions of two simple systems G_1 and G_2 which are connected by a vector channel C_{12}. The buffering size of the asynchronous channel is assumed to be one. In this figure, each transition is labeled by $\vec{u}/\vec{y}$, representing the input and output of the transition. Also for the sim-

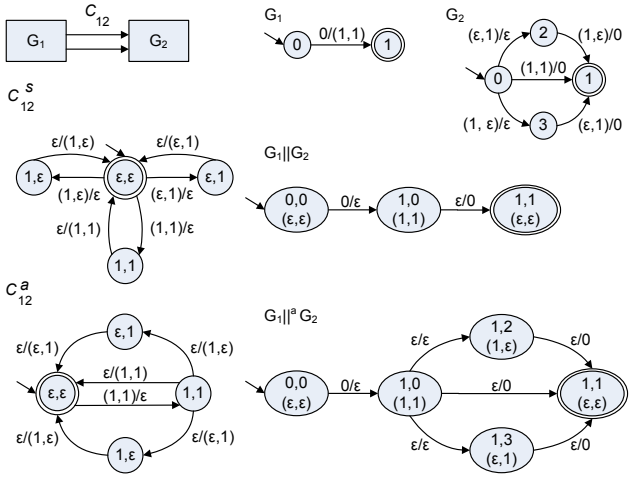

Fig. 3. Simulation relation between $G_1||G_2$ and $G_1||^aG_2$

plicity of illustration, only a partial automaton model of the asynchronous channel C_{12}^a is shown in Figure 3. (This simplification comes from the fact that only "11" can arrive as an input in C_{12}^a.)

It can be verified that the initial state $(0,(\vec{\epsilon},\vec{\epsilon}),0)$ of $G_1||^aG_2$ is simulated by the initial state $(0,(\vec{\epsilon},\vec{\epsilon}),0)$ of $G_1||G_2$, and so the two systems have the same set of accepted responses to a common sequence of inputs. That is, the asynchronous composition $G_1||^aG_2$ preserves the semantics of the synchronous composition $G_1\|G_2$.

Remark 8: Our condition for the correctness of desynchronization, namely the simulation of the asynchronous system by the synchronous system is different from the prior correctness notion of isochrony proposed in [6]. In example 2, the pair of transitions $(0,0/(1,1),0)$ of G_1 and $(0,(\epsilon,1)/\epsilon,2)$ or $(0,(1,\epsilon)/\epsilon,3)$ of G_2 are not synchronizable but not contradictory. It follows that G_1 and G_2 are not isochronous, yet as can be seen by their asynchronous composition, they have the same accepted input-output response.

Remark 9: From Theorem 2, checking the correctness of desynchronization requires checking whether $G^a = \|_i^a G_i$ is simulated by $G^s = \|_i G_i$, which can be done by an existing algorithm for checking a simulation relation. Note when G_i's are finite-state, so is G^s. Also when the communication channels have bounded buffering lengths (which is the practical situation), G^a is also finite-state.

IV. Conclusion

We studied the problem of correctness of desynchronization and presented a framework based on I/O-automata, their compositions and their input/output responses to clearly formulate the notion of correctness: the set of responses to any input sequence is the same in the synchronous design and in the asynchronous implementation. We defined the notion of simulation over I/O-automata and concluded that if asynchronous implementation is simulated by the synchronous specification, then the desynchronization will be correct. That is, under this condition, the asynchronous implementation preserves the semantics while moving from synchrony (design) to asynchrony (implementation) guaranteeing the correctness of desynchronization. This is a new way of characterizing the correctness of desynchronization (as compared to the "iso-/endo-chrony" type conditions proposed in previous works). Under the practical assumption that the communication delay is bounded, the proposed simulation condition is algorithmically verifiable. We illustrated our result through a simple example.

References

[1] G. Berry, A. Bouali, X. Fornari, E. Ledinot, E. Nassor, and R. de Simone, "Esterel: a formal method applied to avionic software development," *Science of Computer Programming*, vol. 36, no. 1, pp. 5–25, 2000.

[2] J. Mikac and P. Caspi, "Flush: A system development tool based on scade/lustre," in *Proceedings of the 10th international workshop on formal methods for industrial critical systems*, Lisbon, Portugal, September 2005.

[3] A. Gamatie and T. Gautier, "Synchronous modeling of avionics applications using the signal language," in *Ninth IEEE Real-time and embedded technology and applications symposium(RTAS'03)*, Washington, DC, USA, May 2003.

[4] H. Hsieh, F. Balarin, L. Lavagno, and A. Sangiovanni, "Synchronous approach to the functional equivalence of embedded system implementations," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 20, no. 8, pp. 1016–1033, 2001.

[5] P. Caspi, A. Girault, and D. Pilaud, "Automatic distribution of reactive systems for asynchronous networks of processors," *IEEE transactions of software engineering*, vol. 25, no. 3, May/June 1999.

[6] A. Benveniste, B. Caillaud, and P. L. Guernic, "From synchrony to asynchrony," in *CONCUR'99: Concurrency Theory, 10th International Conference*, ser. Lecture Notes in Computer Science, J. C. M. Baeten and S.Mauw, Eds. Springer, 1999, vol. 1664, pp. 162–177.

[7] A. Benveniste, P. Caspi, and S. Tripakis, "Distributing synchronous programs on a loosely synchronous, distributed architecture," IRISA, France, Tech. Rep., Oct. 2001.

[8] D. Potop-Butucaru, R. Simone, and J.Talpin, "The synchronous hypothesis and synchronous languages," in *In Embedded Systems Handbook*, R. Zurawski, Ed. CRC Press, 2005, to appear.

[9] D. Potop-Butucaru and B. Caillaud, "Correct-by-construction asynchronous implementation of modular synchronous specifications," in *In Proceedings of the Fifth International Conference on Application of Concurrency to System Design, ACSD*, 2005.

[10] D. Potop-Butucaru, B. Caillaud, and A. Benveniste, "Concurrency in synchronous systems," in *In Proceedings ACSD 2004*, Hamilton,Canada, 2004.

[11] J. Taplin, A. Benveniste, and P. Guernic, "Asynchronous deployment of synchronous transition system," IRISA, Tech. Rep., Oct. 1999.

[12] J. Talpin, "Synchronous modeling and asynchronous deployment of mobile processes," Rapport de Recherche IRISA, No1305, France, Tech. Rep., Mars 2000.

[13] C. Carloni, K.McMillan, and A. Sangiovanni-Vincentelli, "The theory of latency-insensitive design," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 20, no. 8, pp. 1059–1076, 2001.

[14] A. Benveniste, "Non-massive, non-high performance, distributed computing: selected issues," in *Euro-Par 2002. Parallel Processing: 8th International Euro-Par Conference*, Paderborn, Germany, Aug. 2002.

[15] R.Milner, *A calculus of communicating system*, 1980.

[16] R. Kumar, C. Zhou, and S. Basu, "Finite bisimulation of reactive untimed infinite state systems modeled as automata with variables," in *2006 American Control Conference*, Minneapolis, USA, June 2006.

[17] A. Arnold and J. Plaice, *Finite transition systems: semantics of communicating systems*. Hertfordshire, UK: Prentice Hall International (UK) Ltd., 1994.