

Compositional design of isochronous systems

Jean-Pierre Talpin Julien Ouy Loïc Besnard Paul Le Guernic
 INRIA, Unité de Recherche Rennes-Bretagne-Atlantique CNRS, UMR 6074
 IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract

The synchronous modeling paradigm provides strong execution correctness guarantees to embedded system design while making minimal environmental assumptions. In most related frameworks, global execution correctness is achieved by ensuring endochrony: the insensitivity of (logical) time in the system from (real) time in the environment. Interestingly, endochrony can be statically checked, making it fast to ensure design correctness. Unfortunately, endochrony is not preserved by composition, making it difficult to exploit with component-based design concepts in mind. Compositionality can be achieved by weakening the objective of endochrony but at the cost of an exhaustive state-space exploration. This raises a tradeoff between performance and precision. Our aim is to balance it by proposing a formal design methodology that adheres to a weakened global design objective: the non-blocking composition of weakly endochronous processes, while preserving local endochrony objectives. This yields an ad-hoc yet cost-efficient approach to compositional synchronous modeling.

1 Introduction

The synchronous paradigm to embedded system design provides strong execution correctness guarantees while requiring minimal assumptions on the execution environment. In most synchronous formalisms, this is achieved by locally verifying that computation (in the system) is insensitive to communication delays (from the environment), i.e., that the system is endochronous (“time is defined from inside”).

Example Process filter emits x every time the value of its input y changes. Output tags $t_{2,4}$ are timely related to input tags $t_{1..4}$: Process filter is endochronous.

$$y : (t_1,1)(t_2,0)(t_3,0)(t_4,1) \rightarrow \boxed{x = \text{filter}(y)} \rightarrow x : (t_2,1)(t_4,1)$$

In the data-flow formalism Signal, for instance, design is driven by the safety objective of endochrony: endochrony guarantees a synchronization of computations and communications that is independent of possible network latency.

Unfortunately, endochrony is not a compositional property: it is not preserved by synchronous composition.

Example The synchronous composition of filter with an endochronous merge equation (to mean “ d equals if c then y else z ”) is no longer endochronous: timing of the output d is not related to one of the inputs c and y .

$$\begin{array}{l} c : (t_0,0)(t_2,1)(t_4,1) \\ y : \quad (t_2,1)(t_4,1) \\ z : (t_0,1) \end{array} \rightarrow \boxed{d = \text{merge}(c,y,z)} \rightarrow d : (t_0,1)(t_2,1)(t_4,1)$$

In [11], it is shown that compositionality can be achieved by weakening the objective of endochrony: a weakly endochronous system is a deterministic system that can perform independent communications in any order as long as this does not alter its state (i.e. it satisfies the diamond property). It is further shown that the non-blocking composition of weakly endochronous processes is isochronous.

Example The untimed asynchronous composition of processes filter and merge is isochronous: synchronous and asynchronous compositions yield the same flow of values.

$$\begin{array}{l} x : 1001 \\ c : 0110 \\ z : 1010 \end{array} \rightarrow \boxed{x = \text{filter}(y) \parallel d = \text{merge}(c,y,z)} \rightarrow d : 1110$$

However, checking that a system is weakly endochronous requires an exhaustive exploration of its state-space to guarantee that its behavior is independent from the order of inbound communications. This raises a trade-off between performance (incurred by state-space exploration) and flexibility (gained from compositionality). We aim at balancing this trade-off by proposing a formal design methodology that weakens the global design objective (non-blocking composition) and preserves design objectives secured locally (by accepting endochronous components). Our approach consists in globally maintaining a compositional design objective (non-blocking composition) while preserving properties secured locally (endochrony). This yields a less general yet cost-efficient approach to compositional modeling that is able to encompass most of the practical engineering situations. It is particularly aimed at efficiently reusing most of the existing program analysis and compilation algorithms of Signal. To support the present

design methodology, we have designed a simple controller synthesis and code generation scheme [10].

The article starts in Section 2 with an introduction to Signal and its polychronous model of computation. Section 3 defines the necessary analysis framework and Section 4 gives our contribution. We review related works in Section 5 and conclude. A technical report [14] details all formal definitions and proofs outlined in the present article.

2 An introduction to Polychrony

In Signal, a process (written P or Q) consists of the synchronous composition (noted $P|Q$) of equations on signals (written $x = y f z$). A signal x represents an infinite flow of values. It is sampled according to the discrete pace of its clock, noted \hat{x} . An equation $x = y f z$ defines the output signal x by the relation of its input signals y and z through the operator f . A process defines the simultaneous solution of the equations it is composed of.

$$P, Q ::= x = y f z \mid P|Q \mid P/x \quad (\text{process})$$

There are several kinds of equations. A functional equation $x = y f z$ defines an (arithmetic or boolean) relation f between its operands y, z and result x . A delay $x = y \text{ pre } v$ initially defines x by the value v and then by that y had last time it was evaluated. In a delay equation, the signals x and y are synchronous: simultaneously present or simultaneously absent at all times. A sampling $x = y \text{ when } z$ defines x by y when z is true. In a sampling equation, the output signal x is present iff both input signals y and z are present and z holds the value *true*. A merge $x = y \text{ default } z$ defines x by y when y is present and by z otherwise. In a merge equation, the output signal is present iff either of the input signals y or z is present. The process P/x restricts the lexical scope of the signal x to the process P . In the remainder, we write $\mathcal{V}(P)$ for the set of free signal names x of P (they occur in an equation of P and their scope is not restricted). A free signal is an output iff it occurs on the left hand-side of an equation. Otherwise, it is an input signal.

Example We define the process filter depicted in Section 1. It receives a boolean input signal y and produces an output signal x every time the value of the input changes. The local signal z holds the previous value of the input y at all times. When y first arrive, z is initialized to true. If y and z differ then the output x is true, otherwise it is absent.

$$x = \text{filter}(y) \stackrel{\text{def}}{=} (x = \text{true when } (y \neq z) \mid z = y \text{ pre true}) / z$$

Model of computation The formal semantics of Signal in the polychronous model of computation is defined in [7]. Informally, symbolic tags t or u denote periods in time during which execution takes place. Time is defined by a partial order relation \leq on tags ($t \leq u$ means that t occurs before

u). A chain is a totally ordered set of tags and defines the clock of a signal: it samples its values over a series of totally related tags. Events, signals, behaviors and processes are defined as follows:

- an *event* is the pair of a tag $t \in \mathbb{T}$ and a value $v \in \mathbb{V}$
- a *signal* is a function from a *chain* of tags to values
- a *behavior* b is a function from names to signals
- a *process* p is a set of behaviors of same domain
- a *reaction* r is a behavior with one time tag t

Example The meaning of process filter is denoted by a set of behaviors on the signals x and y . Line one, below, we choose a behavior for the input signal y of the equation. Line two defines the meaning of the local signal z by the previous value of y . Notice that it is synchronous to y (it has the same set of tags). Line three, the output signal x is defined at the time tags t_i at which y and z hold different values, as expected in the previous example.

$$\begin{array}{l} y \mapsto (t_1, 1) (t_2, 0) (t_3, 0) (t_4, 1) (t_5, 1) (t_6, 0) \\ z \mapsto (t_1, 1) (t_2, 1) (t_3, 0) (t_4, 0) (t_5, 1) (t_6, 1) \\ x \mapsto \quad (t_2, 1) \quad (t_4, 1) \quad (t_6, 1) \end{array}$$

Notations We use some of the notations formally defined in [14]. We write $\mathcal{T}(s)$ for the chain of tags of a signal s (resp. behavior b , process p) and $\mathcal{V}(b)$ for the domain of a behavior b (resp. process p). The restriction of a behavior b to a set of names X is noted $b|_X$ and its complementary $b|_{\bar{X}}$. Two behaviors b and c are *flow-equivalent*, written $b \approx c$, iff they have the same domain and all signals carry the same values in the same order. Two behaviors b and c are *clock-equivalent*, written $b \sim c$, iff they are equal up to an isomorphism on tags. For instance,

$$\begin{array}{l} \left(\begin{array}{l} y \mapsto (t_1, 1) (t_2, 0) (t_3, 0) \\ x \mapsto \quad (t_2, 1) \end{array} \right) \approx \left(\begin{array}{l} y \mapsto (u_1, 1) (u_2, 0) (u_3, 0) \\ x \mapsto (u_1, 1) \end{array} \right) \\ \left(\begin{array}{l} y \mapsto (t_1, 1) (t_2, 0) (t_3, 0) \\ x \mapsto \quad (t_2, 1) \end{array} \right) \sim \left(\begin{array}{l} y \mapsto (u_1, 1) (u_3, 0) (u_5, 0) \\ x \mapsto \quad (u_3, 1) \end{array} \right) \end{array}$$

We say that two reaction r and s are independent iff they have disjoint domains. Two independent reactions of same time tag t can be merged, written $r \sqcup s$. Two reactions of signal-wise related time tags can be concatenated, written $r \cdot s$, to form a behavior. For instance,

$$\begin{array}{l} (y \mapsto (t_2, 0)) \sqcup (x \mapsto (t_2, 1)) = (y \mapsto (t_2, 0) x \mapsto (t_2, 1)) \\ \left(\begin{array}{l} y \mapsto (t_1, 1) \\ x \mapsto \quad \end{array} \right) \cdot \left(\begin{array}{l} y \mapsto (t_2, 0) \\ x \mapsto (t_2, 1) \end{array} \right) = \left(\begin{array}{l} y \mapsto (t_1, 1) (t_2, 0) \\ x \mapsto \quad (t_2, 1) \end{array} \right) \end{array}$$

Formal properties The formal properties considered in the remainder pertain the insensitivity of timing relations in a process p (its local clock relations) to external communication delays. The property of endochrony, Definition 1, guarantees that the synchronization performed by a process p is independent from latency in the network. Formally, let I be a set of input signals of p , whenever the process p admits two input behaviors $b|_I$ and $c|_I$ that are assumed to be

flow equivalent (timing relations have been altered by the network) then p always reconstructs the same timing relations in b and c (up to clock-equivalence).

Definition 1 A process p is endochronous iff there exists $I \subset \mathcal{V}(p)$ s.t., for all $b, c \in p$, $b|_I \approx c|_I$ implies $b \sim c$.

Example To check that the filter is endochronous, consider two of its possible trace b and c with flow-equivalent input signals $b(y) = (t_1, 1)(t_2, 0)(t_3, 0)(t_4, 1)$ and $c(y) = (u_1, 1)(u_2, 0)(u_3, 0)(u_4, 1)$ (they share no tags, but carry the same flow of values). The filter necessarily constructs the output signals $b(x) = (t_2, 1)(t_4, 1)$ and $c(x) = (u_2, 1)(u_4, 1)$. One notices that b and c are equivalent by a bijection $(t_i \mapsto u_i)_{0 < i < 5}$ on tags: they are clock-equivalent. Hence, the filter is endochronous. This is no longer the case if it is composed with process merge.

Definition 2, below, defines the compositional property of weak endochrony in the polychronous model of computation. Informally, process p is weakly endochronous iff it is deterministic and can perform independent reactions r and s in any order. Note that, by Definition 1, endochrony implies weak-endochrony (e.g. filter is weakly endochronous).

Definition 2 A process p is weakly-endochronous iff

1. p is deterministic: $\exists I \subset \mathcal{V}(p), \forall b, c \in p, b|_I = c|_I \Rightarrow b = c$

2. for all independent reactions r and s , p satisfies:

(a) if $b \cdot r \cdot s \in p$ then $b \cdot s \in p$

(b) if $b \cdot r \in p$ and $b \cdot s \in p$ then $b \cdot (r \sqcup s) \in p$

(c) if $b \cdot (r \sqcup s), b \cdot (r \sqcup t) \in p$ then $b \cdot r \cdot s, b \cdot r \cdot t \in p$

Example For instance, the synchronous composition of processes filter and merge is weakly endochronous: it is deterministic and all combinations of reactions consisting of the signals x, y, z and c belong to its possible behaviors.

In [11], it is proved that weakly endochronous processes p and q are *isochronous* if they are non-blocking (a locally synchronous reaction of p or q yields a globally asynchronous execution $p \parallel q$).

Definition 3 p and q are isochronous iff $p|q \approx p \parallel q$

A process p is non-blocking iff, in any reachable state (characterized by a behavior b), it has a path to a stuttering state (characterized by a reaction r). Notice that the composition of filter and merge is non-blocking.

Definition 4 p is non-blocking iff $\forall b \in p, \exists r, b \cdot r \in p$

3 Formal analysis

For the purpose of program analysis and program transformation, the control-flow tree and the data-flow graph of multi-clocked Signal specifications are constructed. These data structures manipulate clocks and signal names.

Clock and scheduling relations The clock c of a signal denotes a series of instants (a chain of time tags). The clock \hat{x} of a signal x denotes the instants at which the signal x is present. The clock $[x]$ (resp. $[\neg x]$) denotes the instants at which x is present and holds the value true (resp. false). A clock expression e is either the empty clock, noted 0, a signal clock c , or the conjunction $e_1 \wedge e_2$, the disjunction $e_1 \vee e_2$, the symmetric difference $e_1 \setminus e_2$ of e_1 and e_2 .

$c ::= \hat{x} \mid [x] \mid [\neg x]$ (clock)

$e ::= 0 \mid c \mid e_1 \wedge e_2 \mid e_1 \vee e_2 \mid e_1 \setminus e_2$ (clock expression)

$a, b ::= x \mid \hat{x}$ (node)

$R, S ::= c = e \mid a \rightarrow^c b \mid (R|S) \mid R/x$ (timing relation)

Signals and clocks are related by synchronization and scheduling relations, noted R . A scheduling relation $a \rightarrow^c b$ specifies that the calculation of the node b , a signal or a clock, cannot be scheduled before that of the node a when the clock c is present. A clock relation $c = e$ specifies that the signal clock c is present iff the clock expression e is true. Just as ordinary processes P , relations R are subject to composition $R|S$ and to restriction R/x . The semantics of synchronization and scheduling relations in the polychronous model of computation is defined in [7].

Clock inference The inference system $P : R$ associates a process P with its implicit timing relations R . Deduction starts from the assignment of clock relations to primitive equations and is defined by induction on the structure of P : the deduction for composition $P|Q$ and for P/x are induced by the deductions $P : R$ and $Q : S$ for P and Q .

$x = y \text{ pre } v : (\hat{x} = \hat{y})$

$x = y \text{ when } z : (\hat{x} = \hat{y} \wedge [z] \mid y \rightarrow^{\hat{x}} x)$

$x = y \text{ default } z : (\hat{x} = \hat{y} \vee \hat{z} \mid y \rightarrow^{\hat{y}} x \mid z \rightarrow^{\hat{z} \setminus \hat{y}} x)$

$x = y f z : (\hat{x} = \hat{y} = \hat{z} \mid y \rightarrow^{\hat{x}} x \mid z \rightarrow^{\hat{x}} x)$

$P : R \wedge Q : S \Rightarrow P|Q : R|S \quad P : R \Rightarrow P/x : R/x$

In a delay equation $x = y \text{ pre } v$, the input and output signals are synchronous, written $\hat{x} = \hat{y}$, and do not have any scheduling relation. In a sampling equation $x = y \text{ when } z$, the clock of the output signal x is defined by that of \hat{y} and sampled by $[z]$. The input y is scheduled before the output when both \hat{y} and $[z]$ are present, written $y \rightarrow^{\hat{x}} x$. In a merge equation $x = y \text{ default } z$ the output signal x is present if either of the input signals y or z are present. The first input signal y is scheduled before x when it is present, written $y \rightarrow^{\hat{y}} x$. Otherwise z is scheduled before x , written $z \rightarrow^{\hat{z} \setminus \hat{y}} x$. A functional equation $x = y f z$ synchronizes and serializes its input and output signals.

We write $R \models S$ to mean that R satisfies S in the Boolean algebra in which timing relations are expressed: composition $R|S$ stands for conjunction and restriction R/x for existential quantification (some examples are given below). For all boolean signal x in $\mathcal{V}(R)$, we assume that $R \models \hat{x} = [x] \vee [\neg x]$ and $R \models [x] \wedge [\neg x] = 0$.

Example To outline the use of clock and scheduling relation analysis in Signal, we consider the specification and analysis of a one-place buffer. Process buffer implements two functionalities: flip and current. The process flip synchronizes the signals x and y to the true and false values of an alternating boolean signal t . The process current stores the value of an input signal y and loads it into the output signal x upon request.

$$\begin{aligned} x &= \text{buffer}(y) \stackrel{\text{def}}{=} (x = \text{current}(y) \mid \text{flip}(x, y)) \\ \text{flip}(x, y) &\stackrel{\text{def}}{=} (s = t \text{ pre true} \mid t = \text{not } s \mid \hat{x} = [t] \mid \hat{y} = [\neg t]) / st \\ x &= \text{current}(y) \stackrel{\text{def}}{=} \\ &\quad (r = y \text{ default } (r \text{ pre false}) \mid x = r \text{ when } \hat{x} \mid \hat{r} = \hat{x} \vee \hat{y}) / rst \end{aligned}$$

The process buffer has three clock equivalence classes. The clocks \hat{s} , \hat{t} and \hat{r} are synchronous (i.e. $\hat{r} = \hat{s} = \hat{t} = \hat{x} \vee \hat{y}$). Two other synchronization classes, $\hat{x} = [t]$ and $\hat{y} = [\neg t]$, are samples of the signal t .

$$R_{\text{buffer}} \stackrel{\text{def}}{=} \left(\hat{x} = [t] \mid \hat{y} = [\neg t] \mid \hat{r} = \hat{x} \vee \hat{y} \right) / rst$$

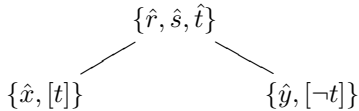
From R_{buffer} , we deduce $\hat{r} = \hat{t}$. Since t is a boolean signal, $\hat{t} = [t] \vee [\neg t]$ (a signal is always true or false when present). By definition of R_{buffer} , $\hat{x} = [t]$ and $\hat{y} = [\neg t]$ (x and y are sampled from t). Hence, we have $\hat{r} = \hat{x} \vee \hat{y}$ and can deduce that $R_{\text{buffer}} \models (\hat{r} = \hat{t})$.

Clock hierarchy and scheduling graph The internal data-structures manipulated by the Signal compiler for program analysis and code generation consist of a clock hierarchy and of a scheduling graph. The clock hierarchy represents the control-flow of a process by a partial order relation. The scheduling graph defines a fine-grained scheduling of otherwise synchronous signals.

Definition 5 The hierarchy \preceq of a process $P : R$ is the transitive closure of the maximal relation defined by the following axioms and rules:

1. for all boolean signals x , $\hat{x} \preceq [x]$ and $\hat{x} \preceq [\neg x]$
2. if $R \models b = c$ then $b \preceq c$ and $c \preceq b$, written $b \sim c$
3. if $R \models b_1 = c_1 \text{ f } c_2$, $f \in \{\wedge, \vee, \setminus\}$, $b_2 \preceq c_1$, $b_2 \preceq c_2$ and b_2 is maximal (i.e. $b_2 = \max_{\preceq} \{b \mid c_1 \succeq b \preceq c_2\}$).

Example The hierarchy of the buffer is constructed by application of the first and second rules of Definition 5. Rule 2 defines three clock equivalence classes $\{\hat{r}, \hat{s}, \hat{t}\}$, $\{\hat{x}, [t]\}$ and $\{\hat{y}, [\neg t]\}$. Rule 1 places the first class above the two others and yields the following structure



A well-formed hierarchy has no relation $b \preceq c$ that contradicts Definition 5. For instance, the hierarchy of the process $x = y$ and $z \mid z = y$ when y is ill-formed, since $\hat{y} \sim [y]$. A process with an ill-formed hierarchy may block.

Definition 6 A hierarchy \preceq is ill-formed iff either $\hat{x} \succeq [x]$ or $\hat{x} \succeq [\neg x]$, for any x , or $b_1 \preceq b_2$ for any $b_1 = c_1 \text{ f } c_2$ such that $c_1 \succeq b_2 \preceq c_2$ and $b_2 \preceq b_1$

A symmetric difference $c \setminus d$ has a disjunctive form if c and d have a common minimum b [2]. Timing relations are in disjunctive form iff they have no clock defined by a symmetric difference relation. For instance, suppose that $d \sim [x]$ and that $c \succeq b \preceq d$. Then, the expression $c \setminus d$ can be eliminated because it can be expressed with $c \wedge [\neg x]$.

Definition 7 A process P of timing R and hierarchy \preceq is well-clocked iff \preceq is well-formed and R is disjunctive.

The transitive closure of scheduling relations in R is the largest relation \rightarrow defined by, 1) if $R \models a \rightarrow^c b$ then $a \rightarrow^c b$, 2) if $a \rightarrow^{e_1} b$ and $a \rightarrow^{e_2} b$ then $a \rightarrow^{e_1 \vee e_2} b$, and 3) if $a \rightarrow^{e_1} c$ and $c \rightarrow^{e_2} b$ then $a \rightarrow^{e_1 \wedge e_2} b$

Definition 8 A process P of timing relations R is acyclic iff the transitive closure \rightarrow of its scheduling relations R satisfy, for all nodes a , if $a \rightarrow^e a$ then $R \models e = 0$.

4 Compositional design criterion

We formulate a decision procedure that uses the clock hierarchy and the scheduling graph of a Signal process to compositionally check the property of isochrony. We start by considering the class of Signal processes P that are reactive and deterministic.

Definition 9 A process P is compilable iff it is acyclic and its relations R are well-clocked.

Property 1 A compilable process P is reactive and deterministic.

Proof An immediate consequence of Prop. 5 in [13].

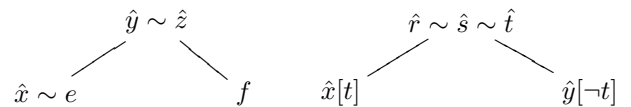
Next, we consider the structure of a compilable Signal specification. It is possibly paced by several, independent, input signals. It necessarily corresponds to a hierarchy \preceq that has several roots. When the hierarchy of a process has a unique root, it is endochronous: the presence of any clock is determined by the presence and values of clocks above it in the hierarchy.

Definition 10 A process p is hierarchic iff its hierarchy has a unique root.

Property 2 A compilable and hierarchic process p is endochronous.

Proof A detailed proof is available in [13].

Example The hierarchies of process filter (Section 1), left, and of the buffer, right, are both hierarchic: they are endochronous. Let $e = ([y] \wedge [\neg z]) \vee ([\neg y] \wedge [z])$ and $f = ([\neg y] \wedge [\neg z]) \vee ([y] \wedge [z])$,



By contrast, a process with several roots necessarily defines concurrent threads of execution. Indeed, and by definition of a hierarchy, its roots cannot be expressed or calculated (or, a fortiori, synchronized or sampled) one with the others. Hence, they naturally define the source of concurrency for the verification of weak endochrony.

Static checking isochrony Checking that a compilable process p is weakly endochronous reduces to proving that the roots of a process hierarchy satisfy property (2) of Definition 2 by using bounded model checking (see [14]). Unfortunately, model-checking is unaffordable for purposes such as program transformation or code generation. In the aim of generating sequential or concurrent code starting from weakly endochronous specifications, we would like to define a simple and cost-efficient criterion to allow for a large and easily identifiable class of weakly endochronous programs to be statically checked and compiled. To this end, we define the following formal design methodology.

Definition 11 *If P is compilable and hierarchic then it is weakly hierarchic. If P and Q are weakly hierarchic, $P|Q$ is well-clocked and acyclic then $P|Q$ is weakly hierarchic.*

By induction on its structure, a process P is weakly hierarchic iff it is compilable and its hierarchy has roots $r_1..n$ such that, for all $1 \leq i < n$, $X_i = \mathcal{V}(\leq^{r_i})$, $P_i = P|_{X_i}$ is weakly hierarchic and the pair $(\prod_{j=1}^i P_j, P_{i+1})$ is well-clocked and acyclic. A proof of Theorem 1 appears in [14].

Theorem 1

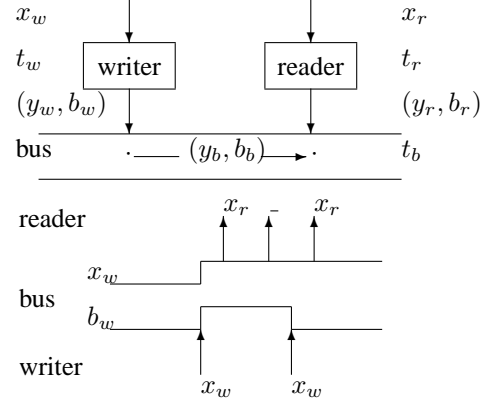
1. A weakly hierarchic process P is weakly endochronous.
2. If P, Q are weakly hierarchic and $P|Q$ is well-clocked and acyclic then P and Q are isochronous.

A compositional design methodology Our static criterion for checking the composition of endochronous processes isochronous defines a cost-effective methodology for the integration of components in the aim of architecture exploration or simulation. Interestingly, this formal methodology meets most of the engineering practice and industrial usage of Signal: the real-time simulation of embedded architectures (e.g. integrated modular avionics) starting from heterogeneous functional blocks (endochronous data-flow functions) and architecture service models.

Example of a loosely time-triggered architecture We consider a simple yet realistic case study build upon the examples we previously presented. We wish to design a simulation model for a loosely time-triggered architecture (LTTA). The LTTA is composed of three devices, a *writer*, a *bus*, and a *reader*. Each device is paced by its own clock.

At the n th clock tick (time $t_w(n)$), the *writer* generates the value $x_w(n)$ and an alternating flag $b_w(n)$. At any time $t_w(n)$, the writer's output buffer (y_w, b_w) contains the last

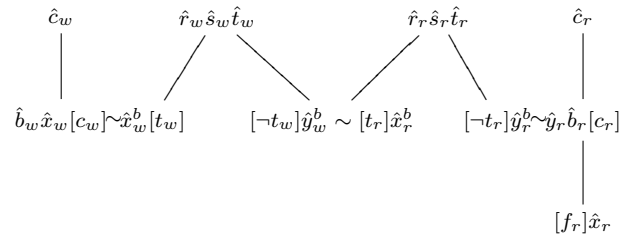
value that was written into it. At $t_b(n)$, the *bus* fetches (y_w, b_w) to store in the input buffer of the reader, denoted by (y_b, b_b) . At $t_r(n)$, the *reader* loads the input buffer (y_b, b_b) into the variables $y_r(n)$ and $b_r(n)$. Then, in a similar manner as for an alternating bit protocol, the reader extracts $y_r(n)$ iff $b_r(n)$ has changed.



A simulation model of the LTTA To model the LTTA in Signal, we consider two data-processing functions that communicate by writing and reading values on an LTT bus. In Signal, we model an interface of these functions that exposes their (limited) control. The writer accepts an input x_w and defines the boolean flag b_w that is carried along with it over the bus. The reader loads its inputs y_r and b_r from the bus and filters x_r upon a switch of b_r . The bus is simulated by a pair of buffers that forward the inputs y_w and b_w to the reader. The clock c_b is not used since the buffers have local clocks. Finally, the process *ltta* is defined by its three components reader, bus and writer.

$$\begin{aligned}
 x_r &= \text{reader}(y_r, b_r, c_r) \stackrel{\text{def}}{=} (x_r = y_r \text{ when filter}(b_r) \mid \hat{y}_r = [c_r]) \\
 (y_w, b_w) &= \text{writer}(x_w, c_w) \stackrel{\text{def}}{=} \\
 &\quad (\hat{x}_w = \hat{b}_w = [c_w] \mid y_w = x_w \mid b_w = \text{not}(b_w \text{ pre true})) \\
 (y_r, b_r) &= \text{bus}(y_w, b_w, c_b) \stackrel{\text{def}}{=} ((y_r, b_r) = \text{buffer}(\text{buffer}(y_w, b_w))) \\
 x_r &= \text{ltta}(x_w, c_w, c_b, c_r) \stackrel{\text{def}}{=} \\
 &\quad (x_r = \text{reader}(\text{bus}(\text{writer}(x_w, c_w), c_b), c_r))
 \end{aligned}$$

We observe that the hierarchy of the LTTA is composed of four trees. Each tree corresponds to an endochronous and separately compiled process, connected to the other at four rendez-vous points (depicted by equivalence relations \sim). The LTTA itself is not endochronous, but it is isochronous because its four components are endochronous and their composition is well-clocked and acyclic.



5 Related Work

In synchronous design formalisms, the design of an embedded architecture is achieved by constructing an endochronous model of the architecture and then by automatically synthesizing ad-hoc synchronization protocols between the elements of this model that will be physically distributed. This technique is called desynchronization and a thorough survey on it is proposed in [8]. In the case of Signal, automated distribution is proposed by Aubry [1]. It consists in partitioning endochronous specifications and synthesizing inter-partition protocols to ensure preservation of endochrony.

In [9], Girault et al. propose a different approach for the synchronous languages Lustre and Esterel. It consists in replicating the generated code of an endochronous specification and in replacing duplicated instructions by inter-partition communications. As it uses notions of bisimulation to safely eliminate blocks, it leads to the construction of a distributed program that consists of endochronously connected programs. But again, distributed code generation is also driven by the global preservation of endochrony. In [11], the so-called property of weak endochrony is proposed. Weak endochrony supports the compositional construction of globally asynchronous system by adhering to a global objective of weak-isochrony. In [12], we propose an analysis of Signal programs to check this property. However, we observe that it is far more costly than necessary, at least for code generation purposes, as it requires an exhaustive state-space exploration. In [6], Dasgupta et al. also propose a technique to synthesize delay-insensitive protocols for synchronous circuits described with Pétri Nets.

In the model of latency-insensitive protocols [4], components are denoted by the notion of *pearl* (“intellectual property under a shell”). A pearl is required to satisfy an invariant of *patience* (which, in turn, implies endochrony [13]) and a *latency-insensitive protocol* wraps the pearl with a generic client-side controller: a so-called relay station. The relay station ensures the functional correctness of the pearl by guaranteeing the preservation of signal flows (i.e. isochrony). It implements this function by suspending the pearl’s incoming traffic as soon as it is reported to exceed its consumption capability. A technique proposed by Casu et al. in [5] refines this protocol to prevent unnecessary traffic suspension by controlling traffic through pre-determined periodic schedules. The latency-insensitive protocol is a compositional approach, and can be seen as a “black-box” approach, in that no knowledge on the pearl (but its capability to be patient) is required. Just as desynchronization, Casu’s variant [5] is a “grey-box” approach, where knowledge on the pearl is needed to synthesize an ad-hoc controller and, at the same time, ensure functional correctness.

6 Conclusions

The clock analysis at the core of our approach shares similarities with both approaches (desynchronization and latency insensitivity). It avoids the need for any explicit suspension mechanism thanks to the determination of precise timing relations. This yields a cost-effective methodology for the compositional design of globally asynchronous architectures starting from synchronous modules. This methodology balances a trade-off between cost (of verification) and compositionality (of design). It maintains a compositional global design objective of isochrony while preserving properties secured locally (endochrony) by checking that composition is non-blocking.

References

- [1] Pascal Aubry. Mises en oeuvre distribuées de programmes synchrones. Thèse de l’Université de Rennes, 1997.
- [2] Loïc Besnard. Compilation de Signal: horloges, dépendances, environnements. Thèse de l’Université de Rennes, 1992.
- [3] Albert Benveniste, Benoit Caillaud, and Paul Le Guernic. Compositionality in dataflow synchronous languages: Specification and distributed code generation. *Information and Computation*, v. 163. Academic Press, 2000.
- [4] Luca Carloni, Ken McMillan, and Alberto Sangiovanni-Vincentelli. The theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 20(9). IEEE, 2001.
- [5] Mario Casu, Luca Macchiarulo. A new approach to latency insensitive design. Design Automation Conference. ACM, 2004.
- [6] Sohini Dasgupta, Dumitru Potop-Butucaru, Benoît Caillaud, Alex Yakovlev. Moving from Weakly Endochronous Systems to Delay-Insensitive Circuits. Formal Methods for GALS Design, Electronic Notes in Theoretical Computer Science. Elsevier, 2006.
- [7] Paul Le Guernic, Jean-Pierre Talpin, and Jean-Christophe Le Lann. Polychrony for system design. *Journal of Circuits, Systems and Computers*. World Scientific, 2003.
- [8] Alain Girault. A survey of automatic distribution methods for synchronous programs. In International Workshop on Synchronous Languages, Applications and Programs. Electronic Notes in Theoretical Computer Science. Elsevier, 2005.
- [9] Alain Girault, Xavier Nicollin, and Marc Pouzet. Automatic rate desynchronization of embedded reactive programs. *ACM Transactions on Embedded Computing Systems*, 5(3), 2006.
- [10] Julien Ouy, Jean-Pierre Talpin, Loïc Besnard, and Paul Le Guernic. Separate compilation of polychronous specifications. *Formal Methods for Globally Asynchronous Locally Synchronous Design*. Electronic Notes in Theoretical Computer Science, Elsevier, 2007.
- [11] Dumitru Potop-Butucaru, Benoit Caillaud, and Albert Benveniste. Concurrency in synchronous systems. In *Formal Methods in System Design*. Kluwer, 2006.
- [12] Jean-Pierre Talpin, Dumitru Potop-Butucaru, Julien Ouy, and Benoit Caillaud. From multi-clocked synchronous specifications to latency-insensitive systems. In *Embedded Software Conference*. ACM, 2005.
- [13] Jean-Pierre Talpin and Paul Le Guernic. An algebraic theory for behavioral modeling and protocol synthesis in system design. Formal Methods in System Design. Special Issue on formal methods for GALS design. Springer, 2006.
- [14] Jean-Pierre Talpin, Julien Ouy, Loïc Besnard, Paul Le Guernic. Compositional design of isochronous systems. Report RR-6227. INRIA, 2007. URL <http://hal.inria.fr/inria-00156499>.