INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

From multi-clock constraints to multi-rate GALS executives

Dumitru Potop-Butucaru — Robert de Simone — Yves Sorel

N° 6021

Novembre 2006

Thème COM

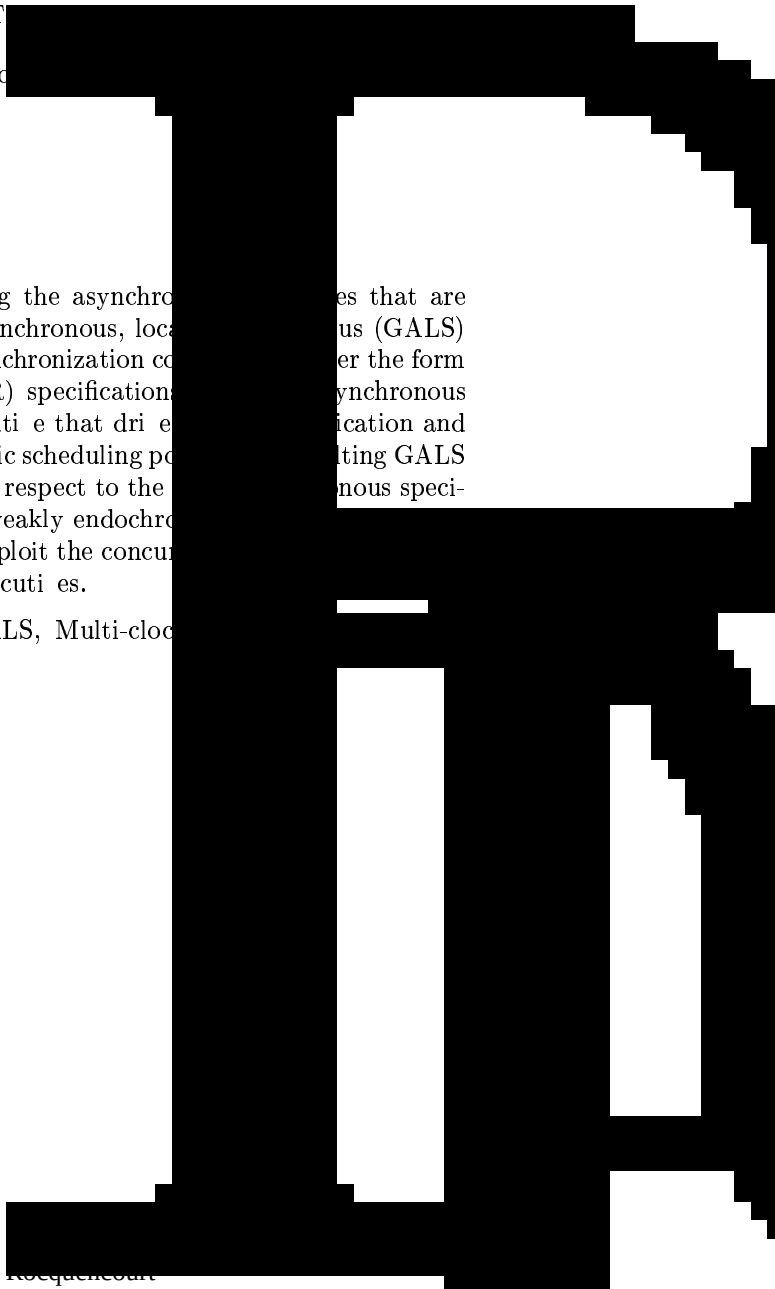
ISSN 0249-6399 ISRN INRIA/RR--6021--FR+ENG

From multi-clock constraints to multi-rate GALS executives

Dumitru Potop-Butucaru , Robert de Simone , Yves Sorel

Thème COM — Systèmes communicants
Projet AOST

Rapport de recherche n° 6021 — No



Abstract: We define a method for synthesizing the asynchro es that are dri ing the synchronous modules of a globally asynchronous, loc us (GALS) system. The technique takes as input high-le el synchronization co er the form of multi-clock modular synchronous reacti e (S/R) specification ynchronous module, our technique produces a multi-rate executi e that dri e ication and clock of the component using a mixed static/dynamic scheduling po lting GALS system is predictable and functionally correct with respect to the nous specification. The approach is based on the theory of weakly endochro on a notion of atomic reaction which allow us to exploit the concu to impro e the communication efficiency of the executi es.

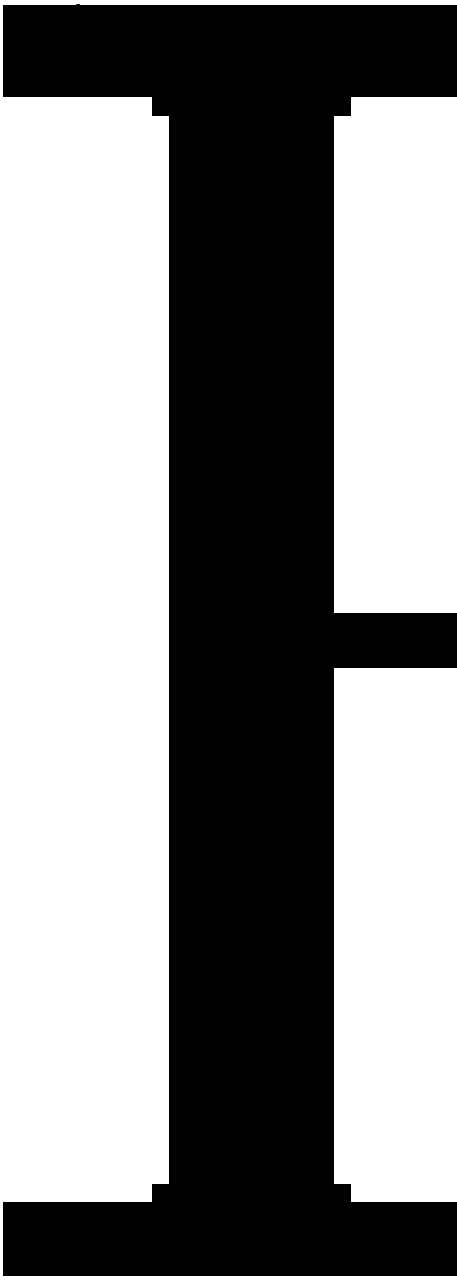
Key-words: Synchronous, Asynchronous, GALS, Multi-cloc dochrony, Automatic distribution

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

Synthèse d'exécutifs GALS à partir de contraintes multi-horloges

Résumé : Nous définissons une méthode pour la synthèse des exécutifs asynchrones qui contrôlent l'exécution des modules synchrones d'un système globalement asynchrone, localement synchrone (GALS). Notre technique prend en entrée des contraintes de synchronisation de haut ni eau, sous la forme de spécifications synchrones réacti es modulaires multi-horloges. Pour chaque module synchrone, notre technique produit un exécutif multi-rythmes qui contrôle les entrées, les sorties, et l'horloge du module en utilisant une politique d'ordonnancement mixte statique/dynamique. Le système GALS résultant est prédictible, correct et complet par rapport à la spécification synchrone d'entrée. L'approche est fondée sur la théorie des systèmes faiblement endochrones (weakly endochronous systems), qui nous permet d'exploiter la concurrence de la spécification pour améliorer l'efficacité des exécutifs.

Mots-clés : Synchrone, Asynchrone, GALS, Multi-horloges, Multi-rythme, faible, Répartition automatique



1 Introduction

De elopment techniques based on synchronous formalisms are today common in arious fields, such as digital circuit design or (safety-critical) embedded software de elopment [3]. Synchronous specifications are used in these fields to model concurrent reacti e systems. The synchronous model allows the explicit representation of the reaction to signal absence, and thus supports a notion of *deterministic concurrency* which facilitates functional modelling and analysis. mplementing synchronous specifications is often hard. This is mainly due to the global synchronization mechanisms of the model, which need to be preser ed, at least in part, to ensure that absence information is not lost when it can influence the beha ior of the system.

n this paper, we address the synthesis of the asynchronous executi es that are dri ing the synchronous modules of a GALS system. The topic draws much attention today, when distributed embedded systems and complex systems-on-chip (SoC) are commonplace in the industry. Our synthesis technique is mathematically founded on the theory of *weakly endochronous systems*, due to Potop, Caillaud, and Ben eniste [16]. Weak endochrony gi es criteria establishing that a synchronous presentation hides a beha ior where the absence information is ne er needed, so that the synchronous specification can safely be executed in an asynchronous en ironment, with predictible results. Weak endochrony extends to a synchronous framework the Mazurkiewicz traces [12]. Thus, the current paper bridges between the fields of synchronous language compilation and classical concurrency theory.

The implementation technique is defined as an *assisted synthesis technique* that takes as input high-le el synchronization constraints structured in a multi-clock synchronous specification. The transformation is performed in two steps. The first one checks whether the specifications of the synchronous modules are weakly endochronous. f they are not, intuiti e diagnostics allow the user to incrementally impro e the signalling protocols. When the specification is weakly endochronous, the second translation step automatically synthesizes the GALS executi es.

Outline. The paper is organized as follows: Section 2 intuiti ely presents our problem, pre ious work, and the desired solution. Section 3 defines the formalism that will support our presentation. Section 4 is on weak endochrony: the original theory of [16] and algorithms to determine if a specification is W[redacted]s are generated for W[redacted] of the resulting GA[redacted]

2 Overview

Our goal is the construction of corr[redacted]ta[redacted]s for GALS systems[redacted] paper, we consider this problem in[redacted]et[redacted]nto account or ensur[redacted] satisfaction of real-time constraint[redacted]he[redacted]ture work.

2.1 Distributed implementation

Our basic model of distributed system is similar to that of Kahn Process Networks (KPN) [10]. A system is formed of computing *components*. The components communicate through *message passing* along *lossless order-preserving asynchronous lines*. For simplicity, our examples only use simple point-to-point communication lines—*asynchronous FIFOs*.

Reading or writing a message on a communication line is blocking and non-interruptible. Blocking reads are part of the basic KPN model. Blocking writes are needed to ensure flow regulation in the absence of timing or synchronization information on the en ironment. With blocking writes, we can use real-life bounded-memory communication lines instead of the unbounded F FOs of the basic Kahn model.

2.2 Modular synchronous specification

To synthesize our executi es, we start with the set of synchronous modules of the GALS system: *off-the-shelf synchronous IPs* or compiled synchronous code, under the form of software *reaction functions*. Following the terminology of [5], we call these implementation modules *pearls*.

[redacted] of its synchronization properties. The [redacted] to the pearls. They should con[redacted]on of the asynchronous executi es contro[redacted]le ant information, such [redacted] operations, can, and should be abs[redacted] to facilitate analysis.

The input of c[redacted]echnique is the modular synchron[redacted]n obtained by putting in par[redacted]le el descriptions of all the pearls.[redacted]sm which will support our p[redacted] small sub-set of the Signal langua[redacted]ans that the specification i[redacted]composition of a number of *processes*.

We use a small[redacted]mple to present our problem, the desire[redacted]e main implementation is[redacted]ample, pictured in Fig. 1, is a simple[redacted] adder, where two indepen[redacted]ord ALUs can be used either indepen[redacted]onized to form a double-[redacted]e choice between synchronized and [redacted] mode is done using the [redacted]o simplify the figure, we compacted bo[redacted]s of each adder under a sin[redacted]e (`I1` and `I2`, respecti ely).

The carry bet[redacted]adders is propagated through the C[redacted]er `SYNC` is present. The two [redacted]LUs are the processes of our sp[redacted] example can be functionally incon[redacted]safely abstract away the ac[redacted]of the adders, and the integer data.

2.2.1 Synchron[redacted]ns

Like any synchron[redacted], we follow a discrete mod[redacted]re executions are sequences of *reac*[redacted] by a *global clock*. Table 1[redacted]le execution of

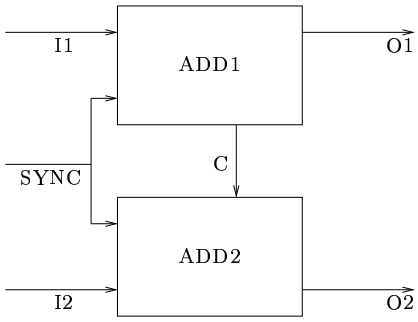


Figure 1: Configurable adder: global dataflow

Clock	1	2	3	4	5	6	7
I1	(1,2)	$\perp$	(9,9)	(9,9)	$\perp$	(2,5)	$\perp$
O1	3	$\perp$	8	8	$\perp$	7	$\perp$
SYNC	$\perp$	$\perp$	$\top$	$\perp$	$\perp$	$\top$	$\perp$
C	$\perp$	$\perp$	1	$\perp$	$\perp$	0	$\perp$
I2	$\perp$	$\perp$	(0,0)	(0,0)	$\perp$	(1,4)	(2,3)
O2	$\perp$	$\perp$	1	0	$\perp$	5	5

Table 1: Sample synchronous run of the example

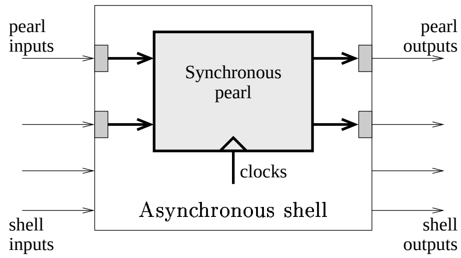


Figure 2: Desired component structure

our example. A reaction is a aluation of the input, output, and internal *signals* of the process. We shall denote with $\mathcal{V}$ the finite set of signals of a process. n our example, $\mathcal{V} = \{ \ 1 \quad 2 \quad YNC\ O1\ O2\ C\}$.

All signals are typed. We denote with $\mathcal{D}_{\mathtt{S}}$ the domain of a signal S. Not all signals need to ha e a alue in a reaction, to model cases where only parts of the process compute. We will say that a signal is *present* in a reaction when it has a alue in $\mathcal{D}_{\mathtt{S}}$. Otherwise, we say that it is *absent*. Absence is simply represented with alue $\perp$, which is appended to all domains $\mathcal{D}_{\mathtt{S}}^{\perp} = \mathcal{D}_{\mathtt{S}} \cup \{\perp\}$. Formally, a reaction of the process is a partial aluation of its signals. We denote with $\mathcal{R}$ the set of all such aluations. The *support* of a reaction r, denoted $s\ pp(r)$, is the set of present signals. For instance, the support of reaction 4 in Table 1 is $\{ \ 1 \quad 2\ O1\ O2\}$.

n many cases we are only interested in the presence or absence of a signal, because it transmits no data, or because we are only interested in synchronization aspects. To represent such signals, the Signal language uses a dedicated `event` type of domain $\mathcal{D}_{\mathtt{event}} = \{\top\}$. n our example, `SYNC` has type `event`.

To represent reactions, we use a *set-like convention* and omit signals with alue $\perp$. n Fig. 1, the signal set is $\{ \ YNC : e\ en\ \ O1\ O2 : \ n\ eger \quad 1 \quad 2 : \ n\ eger\_p \quad r\}$. n Table 1, reaction 4 is denoted $(\ 1^{(9,9)}\ O1^{8} \quad 2^{(0,0)}\ O2^{0})$. The *stuttering reaction* assigning $\perp$ to all signals is denoted $\perp$. n Table 1, reaction 5 is a stuttering reaction.

2.3 Structure of a computing component

To interface between the asynchronous communication lines and the synchronous pearls, we structure our computing components as pictured in Fig. 2.

We are interested here in the asynchronous executi e, also called *shell*, which controls the asynchronous /O and dri es the synchronous pearl of the component. The shell is basically an asynchronous automaton[1]. t takes as input the asynchronous F FOs transmitting the inputs of the pearl, and it can also take additional control inputs added by the synthesis process to ensure correct synchronization. When enough input signals are present, the shell

[1]But its implementation can be synchronous, like in [5], or simply sequential.

I1	(1,2)	(9,9)	(9,9)	(2,5)
O1	3	8	8	7
SYNC		$\top$	$\top$	
C		1	0	
I2	(0,0)	(0,0)	(1,4)	(2,3)
O2	1	0	5	5

Table 2: Corresponding asynchronous run. Correctly reconstructing synchronous inputs from the asynchronous ones is impossible.

triggers a synchronous reaction of the pearl. To do this, the shell pro ides the pearl with the needed inputs, and marks all other pearl inputs as *absent* ($\perp$) e en if alues ha e been recei ed for them. Then, it triggers the computation of the reaction. n software, this amounts to calling the reaction function. n hardware, the pearl clock is enabled. Once the pearl completes its computation: n hardware, the pearl clock is disabled; n software, the state update protocol is performed. The outputs computed by the pearl are transmitted onto the output F FOs. The input alues used in the reaction are acknowledged as read, so that new ones can be accepted from the corresponding F FOs.

2.4 The synthesis problem

We need to synthesize the (hardware or software) shells that will dri e the execution of the the pearls on the components of the architecture.

The main task of the shell is to construct in a consistent way inputs for the synchronous pearl. The input of a reaction must assign a alue, or declare absent, e ery input of the pearl. With this hypothesis, code generation for the pearl is highly simplified[2], for it does not ha e to infer or enforce input presence or absence, nor buffer inputs or outputs to match the asynchronous transmission protocols.

Reconstructing reactions from asynchronous messages must be done in a deterministic fashion, regardless of the message arri al order. This is not always possible. Assume, like in Table 2, that we consider the inputs of Table 1 without the synchronization information. The module `ADD1` will then recei e the first alue (1 2) on the input channel `I1` and $\top$ on `SYNC`. Depending on the arri al order, which cannot be precised, any of the reactions $(\ 1^{(1,2)}\ O1^3\quad YNC^{\top}\ C^0)$ or $(\ 1^{(1,2)}\ O1^3)$ can be triggered, leading to di ergent computations. The problem is that the two reactions are not independent (they do not commute), yet the choice between them cannot be done o er the alue of a message on a gi en channel, so a shell has no means of choosing between them. A similar problem occurs in `ADD2`.

Our approach is to transform the synchronous specification in such a way as to make reaction reconstruction deterministic upto commutation of independent reactions. One such implementation of our example is presented in Fig. 3, with an execution trace in Table 4.

[2]The code can be *exochronous*, in the sense of [14].

Transformation is done by adding communication lines. To discriminate between the non-idependent transitions of `ADD1`, we introduce the Boolean signal `SYNC1`. Value 1 on `SYNC1` indicates that the two adders synchronize, so that `SYNC` and `C` are present. Value 0 indicates an independent computation on `ADD1`. Symmetrically, `SYNC2` is added to `ADD2`. Note that signal `SYNC` becomes useless, as it corresponds to choices o er `SYNC1` and `SYNC1`. Therefore, we can delete it from the dataflow scheme. Note, in Table 4, that the two processes can be now safely run on separate clocks in the GALS implementation.

2.5 Previous solutions

A particular class of solutions to our problem has already been thoroughly studied: The case where e ery pearl reads all inputs and writes all outputs at each reaction (none can be absent). We shall call this the mono-clock case, for reasons that will become clear in Section 3. n this case, implementations ha e been gi en in both software and hardware:

- n SynD [redacted] put of optimized multi-processo [redacted] statically sched-uled on [redacted]
- n Latency- nsensiti e [redacted] [5], the model ser es as basis [redacted] sis of technology-independen [redacted] munication protocols.

Putting our example in mo [redacted] would consist in explicitly transmit [redacted] asynchronous lines the abser [redacted] *i.e.*, transmitting all the symbols in T [redacted] may be inefficient when we [redacted] mize communication, or when we wa [redacted] multi-rate computation.

The first extension towar [redacted] pearls were the *endochronous syster* [redacted] the related hardware *genera* [redacted] *nsensitive systems* [17]. Here, we assu [redacted] presence and absence of all s [redacted] ncrementally inferred starting from t [redacted] from signals that are alway [redacted] ble 3 presents a run of an endochro [redacted] obtained by transforming th [redacted] of our example into one that carrie [redacted] 0 to 3: 0 for `ADD1` executing [redacted] DD2 executing alone, 2 for both ad [redacted] without synchronization (`C` a [redacted] for the synchronized execution [redacted] (`C` present). Note that the [redacted] etermines the presence/abe [redacted] s. The compilation of the Signal la [redacted] ently founded on a e [redacted] ny [1].

The problem with endoc [redacted] ide synchronous modules. The reaction reco [redacted] ess is fully determin [redacted] presence of all signals is synchronized w.r. [redacted] signal(s) in a hierarch [redacted] f we refer to our example, an endochron [redacted] cannot allow concurrency [redacted] pendent computations of the two ad [redacted] cannot allow implementati [redacted] `ADD1` and `ADD2` are implemented [redacted] earl without remo ing all ind [redacted] ween them through added signalli [redacted] gh this is not functionally nece [redacted] leads to o er-synchronizing the in [redacted], and to potential inefficiency. [redacted] t,

[redacted]

Clock	1	2	3	4	5
I1	(1,2)	(9,9)	(9,9)	(2,5)	$\perp$
O1	3	8	8	7	$\perp$
SYNC	0	3	2	3	1
C	$\perp$	1	$\perp$	0	$\perp$
I2	$\perp$	(0,0)	(0,0)	(1,4)	(2,3)
O2	$\perp$	1	0	5	5

Table 3:

Clock1	1			
I1	(1,2)	(9,9)	(9,9)	
O1	3	8	8	
SYNC1	0	1	0	
C		1		
SYNC2		1		0
I2		(0,0)		2,3)
O2		1		5
Clock2		1		4

Table 4: Weakly endochron

endochrony is not compositional, which makes the ncremental de elopment processes difficult.

2.6 Our approach

n this paper, we explain how to allow both synchro synchronized (inde dent) computations to be realized by a single comp approach, the user ca decide at pearl synthesis time (maybe through an e ocess) how to group the processes onto the distributed components, while kn cient shells are automatically generated. For instance, the two processes o can be implemented by separate pearls, or by a single one.

The main contribution is gi en in Section 4. We ex w to determine whether a process is weakly endochronous. This is based on dete imal set of reactions that generate all other reactions. The process is weakly en hen all generators satisfy an *atomicity hypothesis* – they are either asynchronou nable, or non-interferent. When a process is not weakly endochronous, we nee it further by eliminating non-confluent choices that are not determined by a not by absence). Gi en the complexity of this step, which adds supplementar ion lines, we do not fully

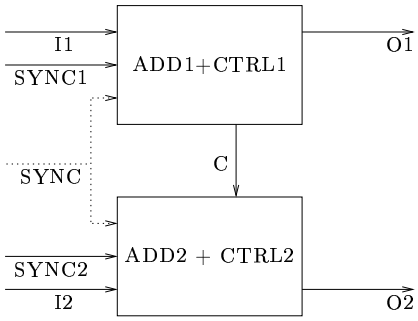


Figure 3: Weakly endochronous solution

automate it. stead of pro iding solutions that might not match the global architecture of the system, our analysis algorithm simply points out through intuiti e error messages why the system is not weakly endochronous. Once the assisted transformation step is completed, the asynchronous shells are automatically generated, as explained in Section 5.

2.6.1 Stateless abstraction

To keep the presentation simple, we present here a simple ersion of the technique, which does not take into account the process state. The approach can be extended to deal with specifications where each process has a state.

3 Synchronous Dataflow in Signal

We use a small sub-set of the high-le el synchronous language Signal to formally present our technique. The Signal multi-clock constraint language allows a simple representation of the synchronization constraints we use. Howe er, the results can be simply applied to any synchronous dataflow specification accompanied by synchronization constraints. n particular, SynD extended using our results.

3.1 Process structure

A Signal process is a *hierarch specification.* Fig. 4 gi es the Sig corresponding to the configural g. 1.

A process is formed of a he ts name, an interface specification, specification, and local declara n our example, the top process i EXAMPLE. ts interface defines s (SYNC

```
process EXAMPLE =
(? event SYNC ; event I1,I2 ;
 ! event O1,O2 ; )
(| (O1,C) := ADD1 (SYNC,I1)
 | O2 := ADD2 (SYNC,I2,C) |)
where boolean C ;

  process ADD1 =
  (? event SYNC ; event I1 ;
   ! event O1 ; event C ; )
  (| I1 ^= O1 | SYNC ^< I1 | C ^= SYNC |) ;

  process ADD2 =
  (? event SYNC ; event I2 ; event C ;
   ! event O2 ; )
  (| I2 ^= O2 | SYNC ^< I2 | C ^=SYNC |) ;
end ;
```

Figure 4: The Signal process of the configurable adder in Fig. 1

its set of input signals, with $\mathcal{O}(P)$ its set of output signals, and with $\mathcal{V}(P)$ the set of all its signals. By definition, $\mathcal{I}(P) \cap \mathcal{O}(P) = \emptyset$ and $\mathcal{I}(P) \cup \mathcal{O}(P) \subseteq \mathcal{V}(P)$.

The dataflow specification of `EXAMPLE` consists of two equations, which define the interconnections between `ADD1`, `ADD2`, and the en ironment. The local definition section defines the internal signal `C`, and the processes `ADD1` and `ADD2`.

Our synthesis technique will only refer directly to the first two le els of the process hierarchy. The unique top-le el process is the specification, and it only contains the global dataflow. ts children are the specifications of the synchronous modules, which can contain more complex beha ioral Signal definitions. Lower-le el processes cannot be distinguished from the specification of the module that contains them.

3.1.1 Finite abstraction

As explained in section Section 2.2, complex data operations that are irrele ant to the synthesis of the shells should be abstracted away in the synchronous specification. For instance, we model the integer inputs and outputs of the two adders with `event` signals. Data abstraction is mandatory for all infinite types (e.g., `integer`, `float`), to make decidable the analysis of the weak endochrony. We say that the synchronous specification is a *finite stateless abstraction* of the corresponding pearl beha ior.

3.2 Dataflow specification

The dataflow specification of a process is formed of *equations* defining *constraints* between the signals of the process. Any reaction satisfying all the equations of a process P is a alid reaction of P. We denote with $\mathcal{R}(P)$ the set of all the reactions of P. Some of the constraints (such as as si n e t) a e f

```
process EXAMPLE_RESULT =
(? boolean SYNC1,SYNC2; event I1,I2 ;
 ! event O1,O2 ; )
(| (O1,C) := CTRL1 (SYNC1,I1)
 | O2 := CTRL2 (SYNC2,I2,C) |)
where boolean C ;

  process CTRL1 =
  (? boolean SYNC1 ; event I1 ;
   ! event O1 ; event C ; )
  (| SYNC1 ^= I1 ^= O1 | C ^= when SYNC1=1
   | (O1,C) := ADD1(when SYNC1=1,I1) |)
  where process ADD1 = ... end ;

  process CTRL2 =
  (? boolean SYNC2 ; event I2 ; event C ;
   ! event O2 ; )
  (| SYNC2 ^= I2 ^= O2 | C ^= when SYNC2=1
   | O2 := ADD2(when SYNC2=1,I2,C) |)
  where process ADD2 = ... end ;
end ;
```

Figure 5: Weakly endochronous process corresponding to Fig. 3. Processes `ADD1` and `ADD2` are copied from Fig. 4.

The operator `default` merges two signals of the same type, gi ing priority to the first. The signal "`X default Y`" is present whene er one of `X` or `Y` is present. t is equal to `X` whene er `X` is present, and is equal to `Y` otherwise.

A Signal process can instantiate other Signal processes inside its dataflow. n Fig. 4 `EXAMPLE` instantiates `ADD1` and `ADD2`. n Fig. 5 `CTRL1` instantiates `ADD1`.

3.3 Notations

A synchronous trace of a process P is any finite succession of reactions of P. The set of traces of P is $\mathcal{T}(P) = \mathcal{R}(P)^*$.

On the alues of any signal , we introduce a partial order, gi en by $\perp \leq$, for all $\in \mathcal{D}_S$. The order on signal alues induces a product partial order $\leq$ on reactions. Note that $r_1 \leq r_2$ if and only if $s_{\blacktriangledown}pp(r_1) \subseteq s_{\blacktriangledown}pp(r_2)$ and $r_1(\ ) = r_2(\ )$ for all $\in s_{\blacktriangledown}pp(r_1)$.

We say of two reactions r_1 and r_2 that they are *non-contradictory*, written $r_1 \quad r_2$, if $r_1(\ ) = r_2(\ )$ for all $\in s_{\blacktriangledown}pp(r_1) \cap s_{\blacktriangledown}pp(r_2)$. Otherwise, we say that the reactions are *contradictory*, written $r_1 \ / \ r_2$. Gi en a set of reactions , we shall say that it is non-contradictory, denoted if any two reactions of are non-contradictory. n this case,

we can define $\vee \quad = \bigvee_{r \in K} r$. Two reactions r_1 and r_2 are *non-interferent* if $s_{\blacktriangledown} pp(r_1) \cap s_{\blacktriangledown} pp(r_2) = \emptyset$.

On non-contradictory reactions, we define the union $\vee$ and intersection $\wedge$ operators as the least upper bound and greatest lower bound induced by $\leq$. We also define the difference $r_1 \setminus r_2$, which has support $s_{\blacktriangledown} pp(r_1) \setminus s_{\blacktriangledown} pp(r_2)$ and equals r_1 on its support.

4 Weak endochrony

4.1 Basic theory

The theory of *weakly endochronous (WE) systems* [16], gi es criteria establishing that a synchronous presentation hides a beha ior that is fundamentally asynchronous and deterministic. Absence information is not needed, which guarantees the deterministic implementability of the synchronous specification in an asynchronous en ironment. Weak endochrony extends to a synchronous framework the Mazurkiewicz traces [12].

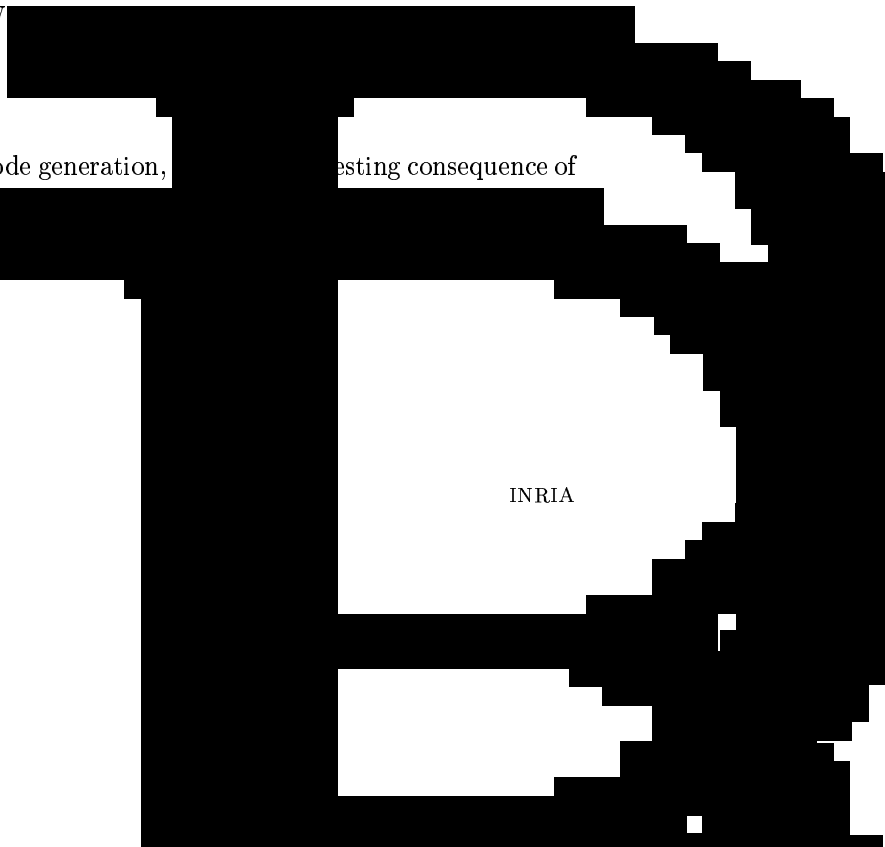
Weak endochrony is defined in an automata-theoretic framework. We simplify it here according to our stateless abstraction:

Definition 1 (stateless weak endochrony) *We say that process P is weakly endochronous if for all $r_1 \ r_2 \in \mathcal{R}(P)$*

$$r_1 \quad r_1 \Rightarrow r_1 \vee r_2 \ r_1 \setminus r_2 \ r_1 \wedge r_2 \ r_2 \setminus r_1 \in \mathcal{R}(P)$$

We already saw, in Section 2.4, that running a synchronous process in an asynchronous en ironment may be a non-deterministic process. Weak endochrony gi es ery general sufficient conditions ensuring that the execution of the process always gi es the same result, regardless of the asynchronous input arri al order.

The intuition behind weak endochrony is that we are looking for systems where (1) all causality is implied by the sequencing of messages on communication channels, and (2) all choices are isible as choices o er the alue (and not present/absent status) of some message. As explained in [15], the axioms of weak endochrony can be traced down to the fundamental result of Keller [11] on the deterministic operation of a system in an asynchronous en ironment. From a different perspecti e, W



similar determinism results.

4.1.1 Atoms

From our point of iew oriented towards code generation, esting consequence of these axioms is that any beha ior of a W

or *atomi*

smallest[3] $\mathcal{R}(P)$ different fr

[3] In the sense of $\leq$.

Atomic transitions ha e ery nice properties. Two atoms $_1$ and $_2$ are either non-interferent (of disjoint support, cf. Section 3.3), or contradictory. Furthermore, any transition of the system can be uniquely decomposed into a set of non-interferent atoms.

Theorem 1 (Generation) *Let P be a weakly endochronous stateless process, and $r \in \mathcal{R}(P)$. Then:*

$$r = \bigvee_{a \in Atoms(P), a \leq r}$$

4.2 Checking weak endochrony

n this section we explain how to check the weak endochrony of the process corresponding to a synchronous module of the GALS system. The analysis is based on the identification of a set of reactions which generate by union all other reactions. The generator set is computed incrementally from the equations of the specification. The process is W
set has the properties of an atom set. f the process is W
will directly ser e in Section 5 to generate the asynchronou

4.2.1 Partial reactions

The domain of signal aluations, defined in Section 2.2.1, allows the repr
plete reactions of processes, as well as the manipulation of reactions of we
processes, where absence information is not needed. Howe er, they do
resentation of absence synchronizations constraints, such as clock exclus
such constraints, we enrich the domain $\mathcal{D}_S^{\perp}$ of each signal with a new
$\mathcal{D}_S^{\perp\!\!\!\perp} = \mathcal{D}_S^{\perp} \cup \{\perp\!\!\!\perp\}$, with $\perp \leq \perp\!\!\!\perp$.

A partial reaction r is any aluation of the signals o er their exten
denote with $\mathcal{R}^{\perp\!\!\!\perp}$ the set of all such aluations. All the operators of Sectio
o er $\mathcal{R}^{\perp\!\!\!\perp}$, according to the new domain structure. The support of a par
set of signals with alue different from $\perp$.

A partial reaction r sets signal to $\perp\!\!\!\perp$ to represent the fact that alu
force to be absent. For instance, if "S^#T" and $r(\ ) \in \mathcal{D}_S$, we need
partial reaction (^{v} $^{\perp\!\!\!\perp}$) cannot be united (using $\vee$) with other partial
is present. This is different from (^{v} $^{\perp}$), which can be composed, for in

Any reaction is a partial reaction, and any partial reaction r has an a
$[r]$ which changes all $\perp\!\!\!\perp$ alues of r into $\perp$ alues. We denote with $\mathcal{R}^{\perp\!\!\!\perp}$
$[r] \in \mathcal{R}(P)\}$

4.2.2 Generators

n this section, we define the non-interferent generator set of a process, a
to compute it. Non-interferent generators of the reactions of a process can
the prime implicants of a logic formula – they are reactions of smallest sup

all other reactions. The main difference is that our algebraic structure is richer than that of the Boolean logic (it has a domain structure), allowing us to exploit concurrency to preser e fewer generators (exponentially fewer, in certain cases). Also, it must be noted that $\perp$ is always a solution, in our case.

Definition 2 (Generator set) *Let P be a process. A set $\mathcal{C} \subseteq \mathcal{R}^{\perp\!\!\!\perp}(P)$ of partial reactions is a generator set of $\mathcal{R}(P)$ if $\mathcal{R}(P) = \{[\bigvee_{r\in K} r] \mid \quad \subseteq \mathcal{C} \wedge \quad \}$.*

Definition 3 (Non-interferent generator set) *Let P be a process. A generator set $\mathcal{C}$ is called a non-interferent generator set of P if:*

$$\forall r_1\ r_2 \in \mathcal{C} : \left.\begin{array}{cc} r_1 & r_2 \\ r_1 \neq r_2 & \end{array}\right\} \Rightarrow s_{\blacktriangledown} pp([r_1]) \cap s_{\blacktriangledown} pp([r_2]) = \emptyset \qquad (\mathcal{NI})$$

Non-interference implies a strong form of minimality for a generator set, also facilitating the manipulation of the generators.

Theorem 2 *Any process has a unique non-interferent generator set. When the process is weakly endochronous*

We also gi e here generators sets of other, non-primiti e equations:

$$\begin{aligned}
\mathcal{G}(P\ \mathtt{X} := \mathtt{Y}) &= \{(X^v\ Y^v) \mid \ \in \mathcal{D}_X \cup \{\perp\!\!\!\perp\}\} \cup \\
&\quad \{(\dot{W}^v) \mid \ \in \mathcal{D}_X\ \ \dot{W} \notin \{X\ Y\}\} \\
\mathcal{G}(P\ \mathtt{X}^{\wedge}{=}\mathtt{Y}) &= \{(X^v\ Y^w) \mid \ \ w \in \mathcal{D}_X\} \cup \{(X^{\perp\!\!\!\perp}\ Y^{\perp\!\!\!\perp})\} \cup \\
&\quad \{(\dot{W}^v) \mid \ \in \mathcal{D}_X\ \ \dot{W} \notin \{X\ Y\}\} \\
\mathcal{G}(P\ \mathtt{X}^{\wedge}\#\mathtt{Y}) &= \{(X^v\ Y^{\perp\!\!\!\perp})\ (X^{\perp\!\!\!\perp}\ Y^v) \mid \ \in \mathcal{D}_X\} \cup \\
&\quad \{(\dot{W}^v) \mid \ \in \mathcal{D}_X\ \ \dot{W} \notin \{X\ Y\}\} \\
\mathcal{G}(P\ \mathtt{X}^{\wedge}\ \ \mathtt{Y}) &= \{(X^v\ Y^w) \mid \ \in \mathcal{D}_X \cup \{\perp\!\!\!\perp\}\ \ w \in \mathcal{D}_Y\} \cup \\
&\quad \{(\dot{W}^v) \mid \ \in \mathcal{D}_X\ \ \dot{W} \notin \{X\ Y\}\}
\end{aligned}$$

Theorem 3 (Merging) *Let P be a process and p and q be sets of equations of P. Then:*

1. *If p is a single equation, then $\mathcal{G}(P\ p)$ defined above give the non-interferent generator set of p in P.*
2. *Procedure 1 (MergeGeneratorSets) computes $\mathcal{G}(P\ p \mid q)$, the unique non-interferent generator set of $p \mid q$ in P.*

Procedure 1 (MergeGeneratorSets) computes in $\mathcal{G}(P\ p \mid q)$ all the minimal reactions of $p \mid q$. t explores e ery minimal combination of compatible atoms in p and

Procedure 1 MergeGeneratorSets

Input: $\mathcal{G}(P\ p)$, $\mathcal{G}(P\ q)$: reaction set
Output: $\mathcal{G}(P\ p \mid q)$: reaction set
  $\mathcal{G}_p \leftarrow \emptyset$; $\mathcal{G}'_p \leftarrow \emptyset$;
  for all $g \in \mathcal{G}(P\ p)$ **do**
    if $[g] = \perp$ **then** $\mathcal{G}'_p \leftarrow \mathcal{G}'_p \cup \{\in\leftarrow\leftarrow\mathcal{TT}\{\mathcal{T}\{\leftarrow\cup$

4.2.3 Weak endochrony check

A process is weakly endochronous as soon as we do not need the forced absence information to distinguish between elements of $\mathcal{G}(P\ p)$.

Theorem 4 (Weak endochrony) *Let P be a process and let p be the set of its equations. Then, P is WE* iff *for all $p\ q \in \mathcal{G}(P\ p)$, $p \neq q$:*

$$\left.\begin{array}{c} s_{\blacktriangledown}pp(p) \cap s_{\blacktriangledown}pp(q) \neq \emptyset \\ [p]\ [q] \neq \bot \end{array}\right\} \Rightarrow \exists\ \ : \mathcal{D}_S \ni p(\ \ ) \neq q(\ \ ) \in \mathcal{D}_S \qquad (\dot{W}E - \ \imath o\ \ s)$$

In this case, the atom set of P is:

$$A\imath o\ \ s(P) = \{[g] \mid g \in \mathcal{G}(P\ p)\ [g] \neq \bot\}$$

Proof sketch: f P is W
reaction, and therefore gen
generators, ha e a contrad
can build the corresponding generator (w
ha e the needed properties.

f $\mathcal{G}(P\ p)$ has property $(\dot{W}E - \ \imath o\ \ s$
and the properties of generators imply we

Checking the weak endochrony of a g
a system is not weakly endochronous, in
f $\in \mathcal{G}(P\ p)$, $\neq$, with $s_{\blacktriangledown}pp(\ ) \cap s$
[ ] and [ ] are reactions of P that are not
asynchronous en ironment. The user mu
contradictory, for instance by addition of
Automatic synthesis of weak endochrony i

4.2.4 Examples

The generator set of the process in Fig. 4

As expected, the process is not weakly endo
and $[(\ 1^{\top}\ O1^{\top}\ \ YNC^{\perp\!\!\!\perp})]$ are neither co

The generator set of the transformed p

$$\{(\ 1^{\top}\ O1^{\top}\ C^{\perp\!\!\!\perp}\ \ YNC2^0)$$
$$(\ 1^{\top}\ O1^{\top}\ \ 2^{\top}\ O2^{\top}\ \ YNC2^1)\}$$

The process is weakly endochronous.

5 Executive generation

Once we determined that a process is weakly endochronous we can generate for it the shell (executi e) described in Section 2. We shall present here a simple technique for generating the shell directly from the atom set.

To further simplify code generation, we also assume here that the weakly endochronous process P describing our shell is /O deterministic.[4] Formally, we shall assume that for any two different $\in A\imath o\ \ s(P)$ we ha e:

$$/ \ \Rightarrow \ \mathcal{I}(P) \ / \ \mathcal{I}(P)$$

where $r\ \mathcal{I}(P)$ is the restriction of r on the input signals.

The executi e we generate for P takes ad antage of atom independence properties. For each atom , independently, we generate a function that cyclically:

1. Awaits the arri al of a new alue ( ) for each signal $\in s_{\blacktriangledown}pp(\ \mathcal{I}(P))$.
2. When all needed input arri ed, trigger a reaction of the pearl, with $\mathcal{I}(P)$ as input, and then output the alues of $\mathcal{O}(P)$.

The actual executi e is formed of all the functions associated with the atoms, running in parallel. n practice, parallelism can be simulated using a simple e ent-dri en approach, where each incoming e ent $^{v}\ \in \mathcal{D}_S$ triggers computations in the functions associated with the atoms ha ing ( ) = .

The actions of step 2 abo e are complex, and must be executed atomically. n a purely asynchronous shell, no two atoms must be executing step 2 at the same time (it is a mutual exclusion region). The operations performed during step 2 are the following:

1. [redacted]
2. [redacted]
3. Trigger the co [redacted] the pearl by enabling the [redacted] re) or calling the reaction fu [redacted] ware).
4. When the reac [redacted] eted: n hardware, disable the cloc [redacted] update the pearl state [redacted]
5. Send the data [redacted] his may block the computation until [redacted] nica- tion lines are f [redacted]
6. Signal the fact [redacted] n $\mathcal{I}(P)$ has been read. Remo e thes [redacted] the await list of ot [redacted] w signals can arri e on the input por [redacted] $P)$).
7. [redacted]

[4] In [redacted] the speci- ficatio [redacted]

6.1 Causality and completeness

n our approach, the causality of computations is determined by the pearl reactions, and by the ability of the shells to control the pearls. n a reaction of the pearl, all input must come before the computation takes place (clock enable/disable cycle in hardware, reaction function call in software), and before any output can be produced. The finest reactions, which gi e the causal dependencies, are the atoms.

By consequence, some reactions of the synchronous model may be unfeasible in the GALS implementation. Consider, for instance, the GALS system synthesized from the following specification:

```
process EXAMPLE2 =
(? event I ; ! event O ;)
(| (A,B) := P1(I) | O:=P2(A,B) |)
where event A,B ;
  process P1 =
  (? event I,A ; ! event B ;) (| A ^= B ^= I |);
  process P2 =
  (? event A ; ! event B,O ;) (| A ^= B ^= O |) ;
end ;
```

dules has exactly one atom: $_1$ = hronous reaction of EXAMPLE2 erform it, because the dependency on that $_1$ is executed e dependency on B requires the in ncy.

We need to e GALS implementation indeed im he functions of the synchronou m. To do this, we need to ensure tha em is acyclic in all possible rations. This problem is similar t compilation problem of oth s languages such as Lustre [9], to the eck of cyclic definitions [13] case of constructi e causality checks

n our case erion ensuring the correctness of the the acyclicity tom system. More complex analyses be the subject k.

6.2 Weak

The weak endo components ensures determinist e syst acyclicity ensu ecified beha iors of the system sible. We ensure now th eha iors are feasible than t More precis any execution synchronous beha i specification.

This is not se. Consider, for ins LS system synthesiz the following s

```
process EXAMPLE2 =
(? boolean I ; ! event O ;)
(| (A,B) := PROD(I) | O:=CONS(A,B) |)
where event A,B ;
  process PROD =
  (? boolean I ; ! event A,B ;)
  (| A:=when I=0 | B:=when I=1 |);
  process CONS =
  (? event A,B ;) (| A^=B |) ;
end ;
```

The atoms of PROD are (^{0} $A^\top$) and (^{1} $B^\top$). The only atom of CONS is ($A^\top$ $B^\top$). The only beha ior of process EXAMPLE2 is $\perp$. Therefore, it has no atom.

Howe er, if PROD and CONS are compiled using the technique of Section 5, the resulting GALS system has non- oid asynchronous beha iors. For instance, after reading ^{0} and ^{1}, it produces $O^\top$. This is due to the fact that the e ents $A^\top$ and $B^\top$ are produced in different reactions of PROD, but read by CONS in a single reaction. The re erse can also happen, where signals produced synchronously are read in different instants.

The criterion ensuring that such stray beha iors cannot occur is *weak isochrony*. After a few notations, we gi e the basic definition of Potop, Caillaud, and Ben eniste [16], as simplified by our stateless framework.

Definition 4 (asynchronous prefixes) *Let P be a process. Then, $e\ d(P)$ is the set of all $\in \mathcal{R}$ such that there exists $\in \mathcal{T}(P)$ with ( ) being the first value on signal in , or $\perp$, if no such value exists.*

n other terms, $e\ d(P_i)$ is the set of all asynchronous prefixes of depth 1 of executions of P. With this notation:

Definition 5 (weak isochrony) *Let P be a process with sub-processes P_1 P_n. We say that the processes P_1 P_n are weakly isochronous if, by definition, any execution of the GALS implementation can be started with a synchronous reaction.*

Formally, for all $r_i \in$ $e\ d(P_i)$, such that any two r_i, r_j are equal on $\mathcal{V}(P_i) \cap \mathcal{V}(P_j)$ we have:

$$\exists \overline{r}_i \in \mathcal{R}(P_i)\ 1 \leq \leq n : \begin{cases} \overline{r}_i \leq r_i \\ \bigvee_{i=1}^{n} \overline{r}_i \in \mathcal{R}(P) \\ \bigvee_{i=1}^{n} \overline{r}_i \neq \perp \end{cases}$$

Checking weak isochrony in its basic form is decidable, but can be ery complex, due to the computation of $e\ d(P_i)$. We propose here a sufficient property adapted to our atom-based approach.

Theorem 5 (sufficient conditions) *Let P be a process with sub-processes P_1 P_n. Then, a sufficient condition for P_1 P_n to be isochronous is that the input of no atom of a component can be covered by atoms of other components, unless these atoms are independent.*

*Formally: We denote with $\mathcal{A}_i$ the union of all A*o *s(P_j) with* ≠ *. Then, P_1 P_n are isochronous if: For all $1 \leq \leq n$, and all* ∈ A*o s(P_i), if $_1$ $_k \in \mathcal{A}_i$ such that for all* ∈ *s*pp(*I*(*P*)) *there exists with $_j$( ) = ( ), then s*pp($_i$) *are mutually disjoint.*

Proof sketch: Consider $r_i \in$ *e* $d(P_i)$ $1 \leq \leq n$ satisfying the hypothesis of Definition 5. There exists and ∈ A*o s(P_i) such that ≤ r_i. Using the hypothesis of the theorem, we can incrementally build a common reaction by appending atoms to one or the other of the P_j. ◇

Note that the criterion offered by Theorem 5 offers a simple computational criterion based on the atoms already computed in pre ious sections. Determining isochrony is done through a simple tra ersal of atom co ers. The criterion is an o er-approximation of weak isochrony. t can be impro ed, for instance by taking into account causality information.

7 Conclusion

We ha e defined a method for synthesizing the asynchronous executi es that are dri ing the synchronous modules of a (GALS) system. The technique takes as input high-le el synchronization constraints. The resulting GALS system is predictable and functionally correct and complete with respect to the initial synchronous specification, and regardless of the size of the communication lines. The technique allows the synthesis of executi es for all specifications whose modules are stateless weakly endochronous.

Future work. The current paper only concerns correctess in an untimed setting. Our long-term objecti e is to take into account and/or guarantee real-time requirements, such as periodicity, end-to-end, or throughput constraints. This in ol es the definition of timing analysis and scheduling techniques compatible with our executi es. On the other hand, the executi es could be simplified under specific timing hypothesis (for instance, the F FO protocols can be simplified if the reader is faster than the writer).

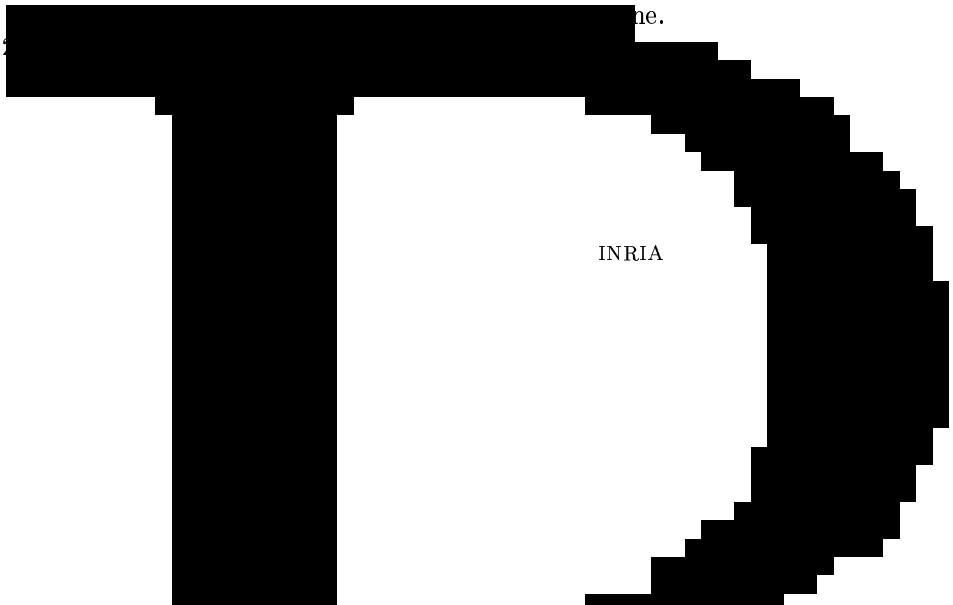
The arious steps of our technique can be impro ed and extended in a ariety of ways, already mentioned in Sections 5 and 6.

References

[1] P. Amagbégnon, L. Besnard, and P. L. Guernic. mplementation of the data-flow synchronous language signal. n *Proceedings PLDI'95*, La Jolla, CA, USA, June 1995.

[2] A. Ben eniste, B. Caillaud, and P. L. Guernic. Compositionality in dataflow synchronous languages: Specification and distributed code generation. *Information and Computation*, 163:125 – 171, 2000.

[3] A. Ben eniste, P. Caspi, S. A. [redacted] ne. The synchronous languages 1[redacted] 2003.



[4] G. Butazzo. *ard real-time computing systems. Predictable scheduling, algorithms and applications.* Kluwer, 2002.

[5] L. Carloni, K. McMillan, and A. Sangio anni-Vincentelli. Theory of latency-insensiti e design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):18, Sep 2001.

[6] P. Caspi and M. Pouzet. Synchronous Kahn networks. n *International Conference on Functional Programming*, pages 226–238, 1996.

[7] T. Grandpierre and . Sorel. From algorithm and architecture specification to auto-

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399