OPEN ACCESS
OPUS AUGSBURG

Universität Augsburg
Universitätsbibliothek

Avoiding irreducible CSC conflicts by internal communication

Mark Schaefer, Walter Vogler, Dominic Wist, Ralf Wollowski

Angaben zur Veröffentlichung / Publication details:

Schaefer, Mark, Walter Vogler, Dominic Wist, and Ralf Wollowski. 2008. "Avoiding irreducible CSC conflicts by internal communication." Augsburg: Universität Augsburg.

Nutzungsbedingungen / Terms of use: **licgercopyright**

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

UNIVERSITÄT AUGSBURG

Avoiding Irreducible CSC Conflicts by Internal Communication

Mark Schaefer
Walter Vogler
Dominic Wist
Ralf Wollowski

Report 2008-02 February 2008

INSTITUT FÜR INFORMATIK
D-86135 AUGSBURG

Copyright © Mark Schaefer, Walter Vogler, Dominic Wist and Ralf Wollowski
Institut für Informatik
Universität Augsburg
D–86135 Augsburg, Germany
http://www.Informatik.Uni-Augsburg.DE
— all rights reserved —

Avoiding Irreducible CSC Conflicts by Internal Communication[*]

Mark Schaefer[1], Walter Vogler[1], Dominic Wist[2] and Ralf Wollowski[2]

[1]Institute of Computer Science
University of Augsburg, Germany
schaefer@informatik.uni-augsburg.de
vogler@informatik.uni-augsburg.de

[2]Hasso-Plattner-Institut
University of Potsdam, Germany
dominic.wist@hpi.uni-potsdam.de
ralf.wollowski@hpi.uni-potsdam.de

Abstract

Resynthesis of handshake specifications obtained e.g. from Balsa or Tangram with speed-independent logic synthesis from STGs is a promising approach [CC06]. To deal with state-space-explosion, we suggested STG decomposition; a problem is that decomposition can lead to irreducible CSC conflicts.

Here, we present a new approach to solve such conflicts by introducing internal communication between the components. We give some first, very encouraging results for very large STGs concerning synthesis time and circuit area.

Keywords: STG, decomposition, state-space-explosion, CSC, handshake, resynthesis

1 Introduction

Asynchronous circuits are a promising type of digital circuits with several well-known advantages compared to their synchronous counterparts (e.g. [vBJN99, SY05]). During the last decade, first entirely asynchronous ICs have already appeared on the market (e.g. [vGvBP+98, KNE+05]).

In order to support asynchronous system design, CAD tools were developed e.g. following syntax-directed translation from a high level HDL such as Balsa [BE00] or based on logic synthesis using graph based specifications such as Signal Transition Graphs (STGs) [Chu87, CKK+02]; the latter are widely used for modelling the behaviour of asynchronous circuits. So far the most successful initiatives are the syntax-directed translation based approaches since they support the designers with a programming language design entry and guarantee a robust implementation of complex specifications. However, the efficiency of the resulting circuits is not satisfactory since the power of boolean minimisation techniques cannot be exploited. To overcome this drawback we are aiming at the incorporation of speed independent (SI) logic synthesis into a design flow based on syntax-directed translation by resynthesis of the system's control path using STGs; see e.g. [CC06] and [WW07b].

With increasing complexity, STG logic synthesis suffers from the state explosion problem. To cope with it, we will apply STG decomposition as in [VW02, VK06].[1] Instead of synthesising an entire overall STG (so-called specification), it will be decomposed into several smaller ones – the component STGs; then for each of them logic synthesis will be applied yielding one circuit per component STG. These interacting component circuits together form the desired asynchronous system. The only disadvantage of the STG decomposition according to Vogler et al. is that it can lead to component STGs which are not SI implementable since they can have so-called

[*]This research was supported by DFG-projects 'STG-Dekomposition' Vo615/8-2 and Wo814/1-3.

[1]There exist further decomposition approaches [CC03, YOM04], but the one used here is proven correct and can be applied to more specifications, in particular to ones without CSC.

irreducible CSC conflicts even if the overall STG has none. In [KS07], it was already attempted to avoid such situations by controlling the signal contraction during the decomposition in such a way that the component STGs satisfy CSC. However, on the one hand this does not always lead to a solution and on the other this could lead to an uncontrollable growth of the component STGs.

Here a new solution to this problem is proposed. After decomposition, we try to 'repair' the irreducible CSC conflicts by introducing internal communication between the components in such a way that an uncontrollable growth is avoided and the overall behaviour of the resulting asynchronous system is preserved.

We have successfully applied the new approach to benchmark examples derived from Balsa components. Although the presented contribution is just a first attempt, we obtained much better results in terms of synthesis time, circuit area and complexity compared to [CC06] (where CSC conflicts are solved in the overall specification using integer linear programming (ILP)).

The next section contains some definitions regarding STGs and their decomposition. The following section introduces some new decomposition operations and related theorems which will be used in the following sections. In Section 4, we present the basic idea for the introduction of internal communication in order to avoid irreducible CSC conflicts in component STGs. An improved algorithmic solution as well as its correctness proof is given in Section 5 and first benchmark results are presented in Section 6. Finally, we give our conclusions in Section 7.

2 Basic Definitions

This section provides the basic notions for Petri nets and STGs, for a more detailed explanation cf. e.g. [CKK+02]. We write $A + B$ instead of $A \cup B$ if $A \cap B = \emptyset$, and $A - B$ instead of $A \setminus B$ if $B \subseteq A$.

2.1 Petri Nets and STGs

A (labelled) *Petri net* is a 6-tuple $N = (P, T, W, M_N, \Sigma, l)$ where P and T are disjoint and finite sets of *places* and *transitions*. $W : P \times T \cup T \times P \to \mathbb{N}_0$ is the *weight function* and M_N the *initial marking*, where a *marking* is a multiset of places, i.e. a function $P \to \mathbb{N}_0$ which assigns a number of *tokens* to each place. The marking of a set of places is defined as the sum of all individual markings. Σ is a set of *actions*, and $l : T \to \Sigma + \{\lambda\}$ is the *labelling function* where λ denotes the empty word. For $A \subseteq T$, we define the *A-labelling* l_A by $l_A(t) = t$ if $t \in A$ and $l_A(t) = \lambda$ otherwise.

A Petri net can be considered as a bipartite graph with weighted and directed edges between places and transitions. If necessary, we write P_N etc. for the components of N or P' (P_i) etc. for the net N' (N_i) etc. Analogous conventions apply later on. For a place p, $N{-}p$ denotes the net in which p and all dependent elements are deleted; for a marking M, $M|_{P'}$ denotes its restriction to $P' \subseteq P$, and $M|_{-p}$ is shorthand for $M|_{P-\{p\}}$.

The *preset* of a place or transition x is denoted as ${}^\bullet x$ and defined by ${}^\bullet x = \{y \in P \cup T \mid W(y, x) > 0\}$, the *postset* of x is denoted as $x^\bullet$ and defined by $x^\bullet = \{y \in P \cup T \mid W(x, y) > 0\}$. These notions are extended to sets as usual. We say that there is an *arc* from each $y \in {}^\bullet x$ to x. A place p is called *marked graph place* or *MG-place* if $\sum_{t \in T} W(t, p) = 1 = \sum_{t \in T} W(p, t)$.

A nonempty sequence $w = x_1 x_2 \dots x_n$ of places and transitions without duplicates is a *path (of N)* if $W(x_i, x_{i+1}) > 0$ for $1 \le i < n$. Obviously, places and transitions have to alternate in a path. With an abuse of notation we often consider a path as the set containing its elements, writing for example $p \in w$. A path w is a *marked graph path* or *MG-path* if every place of w is an MG-place. For a marking M, the *marking* $M(w)$ *of a path* w is defined as $M(w \cap P)$. A path w is called *non-joining, non-forking* resp. if for every transition $t \in w$, $|{}^\bullet t| \le 1$, $|t^\bullet| \le 1$

resp. When joining two paths w_1 and w_2 to $w_1\,w_2$, a possible duplicate element 'in the middle' is deleted implicitly.

A transition t is *enabled under a marking* M if $\forall p \in {}^\bullet t : M(p) \geq W(p,t)$, which is denoted by $M[t\rangle$. An enabled transition can *fire* or *occur* yielding a new marking M', written as $M[t\rangle M'$, if $M[t\rangle$ and $M'(p) = M(p) - W(p,t) + W(t,p)$, for all $p \in P$.

A transition sequence $v = t_1 \dots t_n$ is *enabled under a marking* M (yielding M') if $M[t_1\rangle$ $M_1[t_2\rangle \dots M_{n-1}[t_n\rangle M_n = M'$, and we write $M[v\rangle$, $M[v\rangle M'$ resp.; v is called *firing sequence* if $M_N[v\rangle$. The empty transition sequence λ is enabled under every marking. M is called *reachable* if a transition sequence v with $M_N[v\rangle M$ exists, and $[M_N\rangle$ is the set of all *reachable markings.* A transition is *2-live* if there is a firing sequence for every $n \geq 0$ which contains t n times; a transition is *live* if every reachable marking activates a firing sequence containing t. A net is *2-live*, *live* resp. if each transition is 2-live, live resp.

N is called *bounded* if for every reachable marking M and every place p, $M(p) \leq k$ for some constant $k \in \mathbb{N}$; if $k = 1$, N is called *safe.* N is bounded if and only if the set $[M_N\rangle$ of reachable markings is finite. In this paper, we are only concerned with bounded nets.

We lift the notion of enabledness to transition labels: we write $M[l(t)\rangle\rangle M'$ if $M[t\rangle M'$. This is extended to sequences as usual – deleting λ-labels automatically since λ is the empty word; i.e. $M[a\rangle\rangle M'$ means that a sequence of transitions fires, where one of them is labelled with a while the others (if any) are λ-labelled.

A net has a *dynamic conflict* if there are different transitions t_1 and t_2 such that for some reachable marking M: $M[t_1\rangle$ and $M[t_2\rangle$, but $\exists p \in P : M(p) < W(p,t_1) + W(p,t_2)$. A dynamic conflict implies a *structural conflict*, i.e. ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. The conflict is called an *auto-conflict* if $l(t_1) = l(t_2) \neq \lambda$.

Definition 2.1 ((Transition-)Simulation)
A *simulation from* N_1 *to* N_2 is a relation $\mathcal{S}$ between markings of N_1 and N_2 such that $(M_{N_1}, M_{N_2}) \in \mathcal{S}$ and for all $(M_1, M_2) \in \mathcal{S}$ and $M_1[t\rangle M_1'$ there is some M_2' with $M_2[l_1(t)\rangle\rangle M_2'$ and $(M_1', M_2') \in \mathcal{S}$. A simulation is a *transition-simulation* if it is a simulation when using the labelling l_{T_1} for both N_1 and N_2 in case $T_1 \subseteq T_2$, the labelling l_{T_2} in case $T_2 \subseteq T_1$ resp.

A relation $\mathcal{B}$ is a *bisimulation* between N_1 and N_2 if it is a simulation from N_1 to N_2 and $\mathcal{B}^{-1}$ is a simulation from N_2 to N_1. If such a bisimulation exists, we call the nets *bisimilar.* Transition-bisimulations are defined analogously. $\triangle$

If a simulation exists between N_1 and N_2, then N_2 can go on simulating all actions of N_1 forever; if a bisimulation exists, the nets can work side by side such that in each stage each net can simulate the signals of the other.

Lemma 2.2
Let $\mathcal{S}$ be a transition-(bi)simulation between N_1 and N_2 for the common labelling l'. If $l' = l_{T_1}$, $\mathcal{S}$ is a simulation for any labellings l_1 and l_2 of N_1 and N_2 with $l_1(t) = l_2(t)$ if $t \in T_1$ and $l_2(t) = \lambda$ otherwise. Analogously for $l' = l_{T_2}$.

Proof. Clearly, $(M_{N_1}, M_{N_2}) \in \mathcal{S}$. So let $(M, M') \in \mathcal{S}$. Since $\mathcal{S}$ is a transition simulation, $M[t\rangle M_1$ implies $M'[t\rangle\rangle M_1'$ via $M'[vtv'\rangle M_1'$ with $l'(v) = l'(v') = \lambda$ and $(M_1, M_1') \in \mathcal{S}$. By definition, $l_2(vtv) = l_2(v)l_2(t)l_2(v') = \lambda l_1(t)\lambda = l_1(t)$. $M'[t\rangle M_1'$ implies $M[l_{T_1}(t)\rangle\rangle M_1$ with $(M_1, M_1') \in \mathcal{S}$. If $t \in T_1$, $M[t\rangle\rangle M_1$ via $M[t\rangle M_1$ with $l_1(t) = l_2(t)$ since N_1 has no λ-transitions; otherwise, $M[\lambda\rangle M_1 = M$. $\square$

The *reachability graph* RG_N of a Petri net N is an edge-labelled directed graph on the reachable markings with M_N as root; there is an edge from M to M' labelled $l(t)$ whenever $M[t\rangle M'$. RG_N can be seen as a finite automaton (where all states are accepting). N is *deterministic* if its reachability graph is a deterministic automaton, i.e. if it contains no λ-labelled transitions

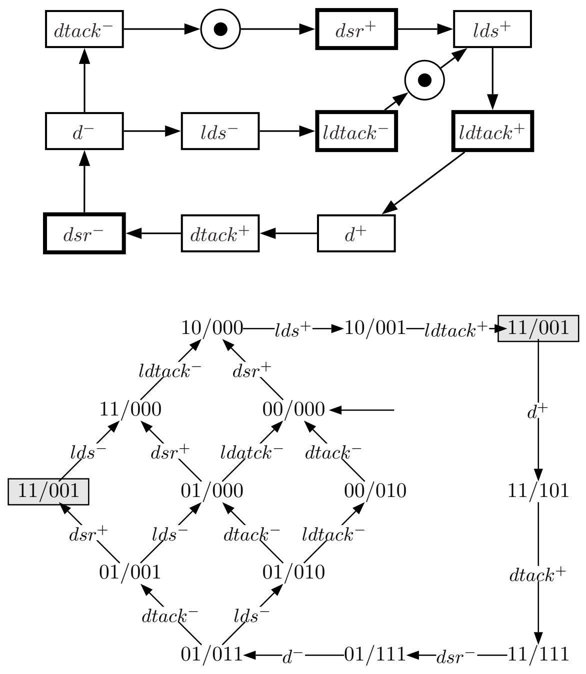

Figure 1: An STG modelling a simplified VME bus controller (top) and its state graph with a CSC conflict between the shaded states (bottom). The signal order in the binary encodings is: *dsr, ldtack, dtack, lds, d.*

and if for each reachable marking M and label $a \in \Sigma$ there is at most one M' with $M[a\rangle\rangle M'$. For deterministic nets, language equivalence and bisimulation coincide.

2.2 Signal Transition Graphs

An *STG* is a tuple $N = (P, T, W, M_N, In, Out, Int, l)$, where $(P, T, W, M_N, Sig^{\pm}, l)$ is a Petri net, In, Out and Int are disjoint sets of *input*, *output* and *internal signals*, and $Sig = In + Out + Int$ is the set of all signals; *signature* refers to this partition of the signals. $Sig^{\pm} = Sig \times \{+, -\}$ is the set of *signal edges* or *signal transitions*; its elements are denoted as s^+, s^- resp. instead of $(s, +)$, $(s, -)$ resp. A plus sign denotes that a signal value changes from *logical low* (written as 0) to *logical high* (written as 1), and a minus sign denotes the opposite direction. We write $s^{\pm}$ if it is not important or unknown which direction takes place; if such a term appears more than once in the same context, it always denotes the same direction. An internal signal is an output of the STGs which cannot be observed by the environment.

To keep the notation short, input/output/internal signal edges are just called input/output/internal edges. Transitions labelled with λ do not correspond to any signal change (cf. state assignment below) and they are also called *dummy-transitions*.

An example of an STG is shown in Fig. 1 (cf. [CKK+02]). Places are drawn as circles containing a number of tokens corresponding to their marking. Unmarked MG-places are not drawn if the incident arcs have the weight 1; they are implicitly given by an arc between the respective transitions. Transitions are drawn as rectangles together with their labelling (input transitions with a thick border), and the weight function is drawn as directed arcs xy whenever $W(x, y) \neq 0$ (and labelled with $W(x, y)$ if $W(x, y) > 1$).

STGs are widely used for specifying the behaviour of *asynchronous circuits*. The idea is as follows: the reachable markings of the STG roughly correspond to the states of the intended

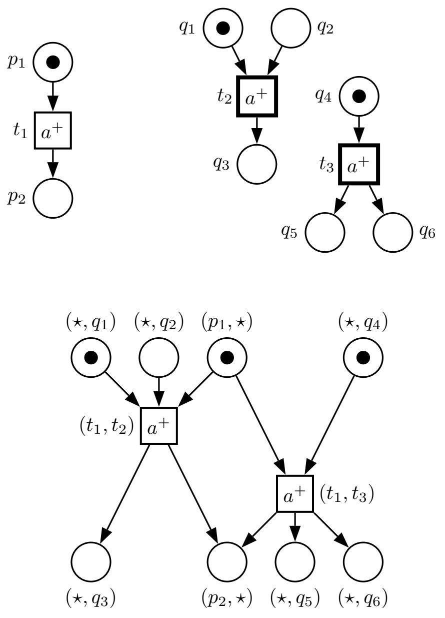

Figure 2: Parallel composition example. The two net fragments in the top line share the signal a, as an output in the left one and as input in the right one. Hence, in their parallel composition (bottom) a is an output.

circuit (viz. the state of all its signals). If some marking activates an output (or internal) signal edge, the circuit must produce the same edge if it is in a corresponding state and the environment of the circuit must be ready to receive it; if some marking activates an input, the environment is allowed to produce it and the circuit must be ready to receive it.

For the first step from markings to circuit states, one defines the notion of *state assignement*: for an STG N, a *state vector* is a function $sv : Sig \rightarrow \{0, 1\}$ where '0' means logical low and '1' logical high. A *state assignment* assigns a state vector sv_M to each marking M of $[M_N\rangle$.

A state assignment must satisfy for every signal $x \in Sig$ and every pair of markings $M, M' \in [M_N\rangle$:

$$\begin{aligned}
&M[x+\rangle\rangle M' \text{ implies } sv_M(x) = 0, sv_{M'}(x) = 1\\
&M[x-\rangle\rangle M' \text{ implies } sv_M(x) = 1, sv_{M'}(x) = 0\\
&M[y^{\pm}\rangle\rangle M' \text{ for } y \neq x \text{ implies } sv_M(x) = sv_{M'}(x)\\
&M[\lambda\rangle\rangle M' \text{ implies } sv_M = sv_{M'}
\end{aligned}$$

If such an assignment exists, it is uniquely defined by these properties[2], and the reachability graph and the underlying STG are *consistent.* From an *inconsistent* STG, one cannot synthesise a circuit, and in this paper we assume that all STGs are consistent. Fig. 1(bottom) shows the reachability graph of the STG in Fig. 1(top); every marking is annotated with its state vector.

We now explain the important concept of *Complete State Coding (CSC).* If there is a state assignment, N has *CSC* if any two reachable markings M_1 and M_2 with the same state vector (i.e. $sv_{M_1} = sv_{M_2}$,) enable the same output and internal signals. Otherwise, N has a *CSC conflict*, cf. e.g. Fig. 1(bottom), and no circuit can be synthesised directly. If CSC is violated, one tries to achieve it by the insertion of internal signals, without changing the external behaviour of the STG (cf. also Definition 2.7 below).

[2]At least for every signal $s \in Sig$ which actually occurs, i.e. $M[s^{\pm}\rangle\rangle$ for some reachable marking M.

Lambdarising a signal, means to change the labelling function such that all transitions corresponding to this signal are labelled with λ and to remove this signal from the signature; *delambdarising* a signal means to restore the former labelling and signature. These operations are important for the decomposition algorithm described below. By contrast, *hiding* a signal set $H \subseteq Out$ from an STG N results in the STG $N/H = (P, T, W, M_N, In, Out - H, Int + H, l)$, i.e. some output signals are now considered to be internal signals.

In the following definition of *parallel composition* $\|$, we will have to consider the distinction between input and output signals. The idea of parallel composition is that the composed systems run in parallel and synchronise on common signals – corresponding to circuits that are connected on the wires corresponding to the signals. Since a system controls its outputs, we cannot allow a signal to be an output of more than one component; input signals, on the other hand, can be shared. An output signal of a component may be an input of other components, and in any case it is an output of the composition. Internal signals of one system must not be used by the other; this is no serious restriction and can always be achieved by a suitable renaming of the respective signals.

A composition can also be ill-defined due to what e.g. Ebergen [Ebe92] calls *computation interference*; this is a semantic problem, and we will not consider it here but later in the definition of correctness.

The parallel composition of STGs N_1 and N_2 is defined if $Out_1 \cap Out_2 = Int_1 \cap Sig_2 = Int_2 \cap Sig_1 = \emptyset$. The place set of the composition is the disjoint union of the place sets of the components; therefore, we can consider markings of the composition (regarded as multisets) as the disjoint union of markings of the components. To define the transitions, let $A = Sig_1 \cap Sig_2$ be the set of common signals. If e.g. s is an output of N_1 and an input of N_2, then an occurrence of an edge $s^\pm$ in N_1 is 'seen' by N_2, i.e. it must be accompanied by an occurrence of $s^\pm$ in N_2. Since we do not know a priori which $s^\pm$-labelled transition of N_2 will occur together with some $s^\pm$-labelled transition of N_1, we have to allow for each possible pairing. Thus, the *parallel composition* $N = N_1 \parallel N_2$ is obtained from the disjoint union of N_1 and N_2 by combining each $s^\pm$-labelled transition t_1 of N_1 with each $s^\pm$-labelled transition t_2 from N_2 if $s \in A$. Such transitions are pairs and the firing $(M_1, M_2)[(t_1, t_2)\rangle(M'_1, M'_2)$ of N corresponds to the firings $M_i[t_i\rangle M'_i$ in N_i, $i = 1, 2$; for an example of a parallel composition, see Fig. 2. More generally, we have $(M_1, M_2)[w\rangle\rangle(M'_1, M'_2)$ iff $M_i[w|_{N_i}\rangle\rangle M'_i$ for $i \in \{1, 2\}$, where $w|_{N_i}$ denotes the projection of the trace w onto the signals of the STG N_i.

It is easy to see that N is deterministic if N_1 and N_2 are. However, as illustrated in Fig. 2, N might have structural auto-conflicts even if none of the N_i has them.

Obviously, we can define the parallel composition of a finite family (or collection) $(C_i)_{i\in I}$ of STGs as $\|_{i\in I}\, C_i$, provided that no signal is an output signal of more than one of the C_i.

We now introduce transition contraction (see e.g. [And83] for an early reference), which will be most important in our decomposition procedure. We essentially repeat from [VK06], where further discussions can be found.

Definition 2.3 (Transition Contraction)
Let N be a Petri net and $t \in T$ with $l(t) = \lambda$, ${}^\bullet t \cap t^\bullet = \emptyset$ and $W(p, t), W(t, p) \leq 1$ for all $p \in P$. We define the *t-contraction* N' of N by

$$\begin{array}{rcl}
P' & = & \{(p, \star) \mid p \in P - ({}^\bullet t \cup t^\bullet)\} \\
 & & \cup\ \{(p, p') \mid p \in {}^\bullet t, p' \in t^\bullet\} \\
T' & = & T - \{t\} \\
W'((p, p'), t_1) & = & W(p, t_1) + W(p', t_1) \\
W'(t_1, (p, p')) & = & W(t_1, p) + W(t_1, p') \\
l' & = & l|_{T'} \\
M_{N'}((p, p')) & = & M_N(p) + M_N(p')
\end{array}$$

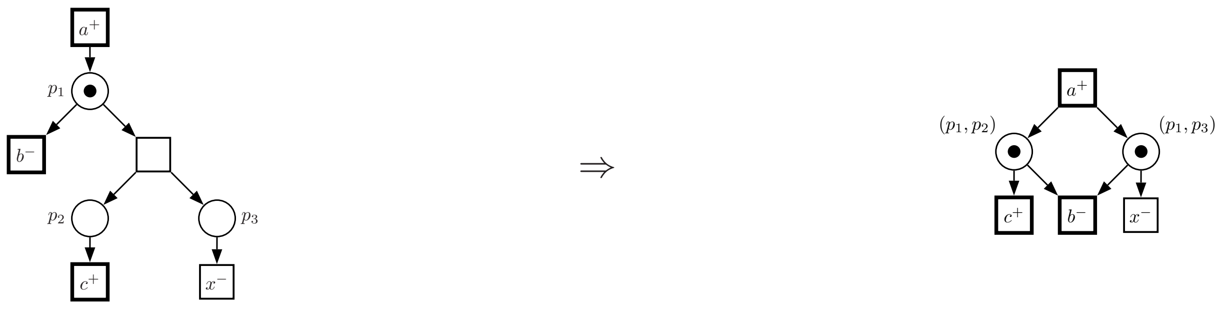

Figure 3: Example of a transition contraction in an STG.

$$In' = In \quad Out' = Out \quad Int' = Int$$

In this definition, $\star \notin P \cup T$ is a dummy element; we assume $W(\star, t_1) = W(t_1, \star) = M_N(\star) = 0$.

We say that the markings M of N and M' of N' satisfy the *marking equality* if for all $(p, p') \in P'$

$$M'((p, p')) = M(p) + M(p').$$

For two different transitions t_1, t_2 with $t_1 \neq t \neq t_2$, we call the unordered pair $\{t_1, t_2\}$ a *new conflict pair* whenever ${}^\bullet t \cap {}^\bullet t_1 \neq \emptyset$ and $t^\bullet \cap {}^\bullet t_2 \neq \emptyset$ in N (or vice versa); if $l(t_1) = l(t_2) \neq \lambda$, we speak of a *new structural auto-conflict.*

A transition contraction is called *secure* if either $({}^\bullet t)^\bullet \subseteq \{t\}$ (*type-1 secure*) or ${}^\bullet(t^\bullet) = \{t\}$ and $M_N(p) = 0$ for some $p \in t^\bullet$ (*type-2 secure*). $\triangle$

Note that, in general, N' might fail to be consistent, even if N is; but secure contractions preserve consistency (see [VK06]).

Fig. 3 shows a part of a net and the result of contracting the λ-transition, where the b^-- and the c^+-labelled transition form a new conflict pair; note that this is also true, if they already had a common place (not drawn) in their presets in N – they now have a new such place.

We conclude this section by defining redundant transitions and implicit places; the deletion of such a transition, place resp., (including the incident arcs) is another transformation that can be used in our decomposition algorithm.

A transition t is *redundant* if either it is a λ-transition with $W(p, t) = W(t, p)$ for each place p (i.e. t is a *loop-only* transition), or there is another transition t' with the same label such that $W(p, t) = W(p, t')$ and $W(t, p) = W(t', p)$ for each place p (i.e. t is a *duplicate* transition).

A place p is *implicit* if it can be deleted from the net without changing the set of firing sequences. However, detecting implicit places is NP-hard and during decomposition only *redundant places* [Ber87] are deleted. Redundant places are implicit (but in general not vice versa); they are defined on the structure of the net and there are LP techniques to find them. However, these techniques are actually not efficient enough and only the subset of *shortcut places* [SVJ05] is deleted. An MG-place p is a shortcut place if there is an MG-path w between $t \in {}^\bullet p$ and $t' \in p^\bullet$ with $M_N(p) \geq M_N(w)$. In [SVJ05] it was shown that shortcut places are indeed redundant. They will be used in the correctness proofs below. The following proposition is essentially from [VK06, KS07].

Proposition 2.4
A place p of N is implicit if and only if for all reachable markings M of N, $M|_{-p}[t\rangle \Rightarrow M[t\rangle$. If N' is obtained from N by deleting p, then $\mathcal{B} = \{(M, M|_{-p}) \mid M \in [M_N\rangle\}$ is a (transition-)bisimulation between N and N'.

Proposition 2.5
Let N' be obtained from N by insertion of an implicit place p' (i.e. p' is implicit in N'). If p is implicit in N, then
(1) p is implicit in N' and
(2) p' is implicit in N'–p.
If a transition t is redundant in N and not adjacent to p in N', then
(3) p is implicit in $N'-t$ and
(4) t is redundant in N'.

Proof. We check the condition of Proposition 2.4.

(1) If $M' \in [M_{N'}\rangle$, then $M'|_{-p}[t\rangle \Rightarrow M'|_{-p,p'}[t\rangle$ for $t \in T$. Since p is implicit in $N = N'$–p' and p' in N', this implies $M'[t\rangle$.

(2) Each reachable marking of N'–p has the form $M'|_{-p}$ for $M' \in [M_{N'}\rangle$, which follows with (1) from the transition bisimulation of Proposition 2.4. If $M'|_{-p,p'}[t\rangle$ in N'–p, then $M'[t'\rangle$ as above and thus $M'|_{-p}[t\rangle$.

(3) By Proposition 2.4, deleting transitions does not affect the implicitness of a place.

(4) Redundancy is structurally defined and the new place p does not change the neighbourhood of t, hence the claim follows. □

Lemma 2.6
In a net N, let p be an implicit MG-graph place with ${}^\bullet p = \{t_1\}$ and $p^\bullet = \{t_2\}$. If t_2 is 2-live, there is a path from t_1 to t_2 in N–p.

Proof. (Sketch) Assume otherwise. Consider a firing sequence w in N–p containing t_2 $M_N(p)+1$ times (since t_2 is 2-live such a sequence exists); consider the corresponding Petri net process π (cf. e.g. [Rei85]). We can remove from π all events (with their postsets) that do not have a path (in π) to a t_2-labelled event, resulting in a new process π' and a corresponding firing sequence w' which also contain t_2 $M_N(p)+1$ times.

Since paths in π correspond to paths in N–p (via the labelling of π), neither π' nor w' do contain t_1. Hence, t_2 can fire $M_N(p)+1$ times in N–p without firing t_1. This is not possible in N, a contradiction. □

2.3 STG Decomposition

Now, the STG decomposition algorithm from [VW02, VK06] is roughly described; most important are transition contractions and (dynamic) auto-conflicts.

Synthesis with STG decomposition works roughly as follows: a partition of the output signals of the given *specification* STG N is chosen, and the decomposition algorithm decomposes N into component STGs, one for each set in this partition. Then, from each component equations for the corresponding outputs are derived from the respective reachability graph, instead of deriving the equations from N itself.

Very often, adding up the sizes of these graphs gives a number much smaller than the size of the reachability graph of N, in which case the decomposition can be seen as successful. Actually, it might already be beneficial if each reachability graph is smaller than the one of N, in particular for reducing peak memory.

Of course, the behaviour of the specification should be preserved in some sense; this is captured by a variety of bisimulation, tailored to the specific needs of asynchronous circuits:

Definition 2.7 (Correct Decomposition)
A collection of deterministic components $(C_i)_{i\in I}$ is a *correct decomposition* of (or simply *correct w.r.t.*) a deterministic STG N – also called *specification – when hiding H*, if $C = (||_{i\in I}C_i)/H$ is

defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is an *STG-bisimulation* $\mathcal{B}$ between the markings of N and those of C with the following properties:

$(M_N, M_C) \in \mathcal{B}$ and for all $(M, M') \in \mathcal{B}$, we have:

(N1) If $a \in In_N$ and $M[a\pm\rangle\rangle M_1$, then either $a \in In_C$, $M'[a\pm\rangle\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M'_1 or $a \notin In_C$ and $(M_1, M') \in \mathcal{B}$.

(N2) If $x \in Out_N$ and $M[x\pm\rangle\rangle M_1$, then $M'[vx\pm\rangle\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M'_1 with $v \in (Int_C\pm)^*$.

(N3) If $u \in Int_N$ and $M[u\pm\rangle\rangle M_1$, then $M'[v\rangle\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M'_1 and $v \in (Int_C\pm)^*$.

(C1) If $x \in Out_C$ and $M'[x\pm\rangle\rangle M'_1$, then $M[vx\pm\rangle\rangle M_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M_1 with $v \in (Int_N\pm)^*$.

(C2) If $x \in Out_i$ for some $i \in I$ and $M'|_{P_i}[x\pm\rangle\rangle$, then $M'[x\pm\rangle\rangle$. (no *computation interference*)

(C3) If $u \in Int_C$ and $M'[u\pm\rangle\rangle M'_1$, then $M[v\rangle\rangle M_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M_1 and $v \in (Int_N\pm)^*$.

Here, and whenever we have a collection $(C_i)_{i\in I}$, P_i stands for P_{C_i}, Out_i for Out_{C_i} etc. $\triangle$

In a simple case, $(C_i)_{i\in I}$ consists of just one component C_1 (immediately implying (C2)).

(C2) ensures that no computation interference occurs, i.e. if a component produces an output (which is under the control of this component), then the other components expect this signal if it belongs to their inputs, and no malfunction of these other components must be feared. (C2) is actually also satisfied for $x \in Int_i$, since internal signals of one component are by definition unknown to the other components.

For STGs the notion of *speed-independence (SI)* [CKK$^+$02] is important: an STG is speed-independent if the intended circuit works correctly under arbitrary delays of the gates (while the signal propagation is considered instantaneous). As a consequence, an STG has to be *input proper*, i.e. no input becomes enabled by an internal signal, since otherwise the environment might produce the input before the internal signal is produced by the circuit. Alternativly, one could drop the SI requirement by making timing assumptions, e.g. that an internal signal is always faster than an input. In this alternative, the enabling of an output by an internal signal should not be interpreted as a causal but a temporal relation.[3]

In [SV07] it was shown that the above correctness notion implies that the implementation is input proper; in particular, if the solution of a CSC-conflict inserts an internal signal in front of an input, it is not correct in the sense of Defintion 2.7.

The following theorem repeats from [SV07].

Theorem 2.8 (Hierarchical Decomposition)
Let N be an STG and $(C_i)_{i\in I}$ a correct decomposition of N when hiding H_C.

(1) If $(C_k)_{k\in K}$ is a correct decomposition of some C_j when hiding H_K ($j \in I$, $I \cap K = \emptyset$), then $(C_i)_{i\in I'}$ with $I' := I + K - \{j\}$ is a correct decomposition of N when hiding $H_C \cup H_K$ if $\bigcup_{k\in K} Out_{C_k} \setminus H_K = Out_{C_j}$ and $(\bigcup_{k\in K} Int_{C_k} \cup H_K) \cap \bigcup_{i\in I\setminus\{j\}} Sig_{C_i} = \emptyset$.

(2) $(C_i)_{i\in I}$ is a correct decomposition of N/H when hiding $H_C \cup H$.

We now discuss our decomposition algorithm in more detail. In the following, we assume that we are given a deterministic, consistent specification N without internal signals.[4]

First, one chooses a *feasible partition*, i.e. a family $(In_i, Out_i)_{i\in I}$ for some set I such that the sets Out_i are a partition of Out, $In_i \subseteq Sig \setminus Out_i$ for each i and furthermore:

[3]This can be modelled by a so-called tcb-concurrency [WB00].

[4]For the decomposition algorithm, internal signals can be considered as outputs; see [SV07] for more details.

- If two output signals x_1, x_2 are in structural conflict in N, then they have to be in the same Out_i.
- If there are $t, t' \in T$ with $t' \in (t^\bullet)^\bullet$ (t is called *syntactical trigger of* t'), then $l(t') \in Out_i$ implies $l(t) \in In_i \cup Out_i$.

If we have a feasible partition, we can build another feasible one by adding additional input signals to one of the members.

For each member (In_i, Out_i) of the partition, an *initial component* is generated from N: in a copy of the original STG N, every signal not in $In_i \cup Out_i$ is lambdarised and the signals in In_i are considered as inputs of this component – even if they are outputs of N. Then the following three reduction operations are applied to an initial component until no more λ-labelled transitions remain:

- secure contraction of a λ-labelled transition
- deletion of an implicit place
- deletion of a redundant transition

Unfortunately, it is not always possible to contract all λ-transitions. Besides the technical cases where the the contraction is not defined or not secure (possibly leading to an incorrect decomposition), the contraction might also generate a new auto-conflict. The latter reveals non-determinism which is present in the respective initial component but not in the specification. This indicates that the component has not enough information to properly produce its outputs. Such a contraction is disallowed and consequently a new signal is added as follows.

If λ-transitions remain, *backtracking* is applied, i.e. a new input is added to the component. Technically, this input is added to the initial partition and the new corresponding initial component is derived and reduced from the beginning. The new input signal is taken from the former label of a non-contractible λ-transition. As discussed above, the new partition is feasible again. This cycle of reduction and backtracking is repeated till all λ-transitions of the initial component can be contracted.

In principle every so-called *totally admissible operation* [VK06] can be used for reduction. The next proposition taken from [VK06] gives a sufficient condition for an operation to be totally admissible, and in the next section a new operation fulfilling this condition is presented.

Proposition 2.9 (Totally Admissible Operation)
An operation is admissible *if, whenever applied to an STG satisfying (a) and (b) below, it results in a bisimilar STG satisfying these properties again:*

(a) There is no structural λ/output conflict, i.e. between a λ-transition and one labelled with an output.

(b) No λ-transition is a syntactical trigger of an output-transition.

The operation is totally admissible *if additionally it decreases the termination function (sc, tc, pc) (with lexicographical order), where sc is the number of lambdarised signals (w.r.t some STG N), tc and pc are the numbers of transitions and places.*

3 Place Refinement and Subnet Contraction

In this section, we introduce some new operations, which are shown to be compatible with decomposition, reduction resp. In particular, *gyroscope insertion* inserts essentially what is known as toggle-transition.

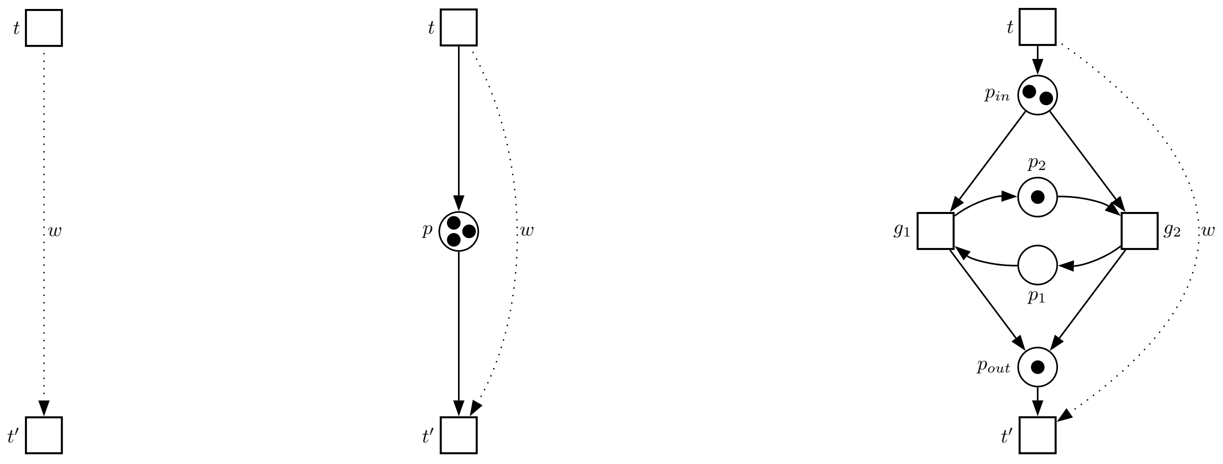

Figure 4: Gyroscope insertion between t and t' with initial marking $(2, 1)$ via place insertion (here: a shortcut place due to w) and place refinement.

Definition 3.1 (Place-refinement, subnet-contraction and gyroscope insertion)

Let N be an STG.
(1) For a place $p \in P$, consider a net N' with:

- $P' = P - \{p\} + \{p_{in}, p_{out}, p_1, p_2\}$
- $T' = T + \{g_1, g_2\}$
- $W'(x, y) = W(x, y)$ if $x, y \in P - \{p\} + T$
  $W'(t, p_{in}) = W(t, p)$ for $t \in T$
  $W'(p_{out}, t) = W(p, t)$ for $t \in T$
  $W'(p_i, g_i) = 1 = W'(g_i, p_{3-i})$, $i = 1, 2$
  $W'(p_{in}, g_i) = 1 = W'(g_i, p_{out})$, $i = 1, 2$
- $M_{N'}(p') = M_N(p')$ for $p' \in P - \{p\}$
  $M_{N'}(p_1) = 1$, $M_{N'}(p_2) = 0$
  $(M_{N'}(p_{in}), M_{N'}(p_{out})) = (in, out)$ with $in + out = M_N(p)$

Starting from N, N' is called a *place-refinement of* p *with initial marking* (in, out); the labelling of the new transitions and the signature of N' can be chosen arbitrarily. Starting from N', N is called a *subnet-contraction* if g_1 and g_2 are λ-transitions.
(2) A *gyroscope insertion* with initial marking (in, out) inserts a new implicit place $p \notin P$ with $in + out$ tokens into N (giving the intermediate N'') and applies place refinement with initial marking (in, out) to it (giving N'). A *gyroscope insertion between* t and t' $(t, t' \in T)$ with initial marking (in, out) inserts p between t and t' (i.e. ${}^\bullet p = \{t\}$ and $p^\bullet = \{t'\}$ with arc weights 1).

A gyroscope insertion is called an *input/output/internal gyroscope insertion* if g_1 and g_2 are labelled in N' with s^+, s^- resp. and s is a fresh input, output or internal signal resp.; it is called a *dummy gyroscope insertion* if g_1 and g_2 are labelled with λ. $\triangle$

Subnet-contraction is used for the correctness proofs below and not really intended as reduction operation. But in principle, one could try to apply it if only backtracking is the alternative, though the odds for this to succeed seem to be low.

Proposition 3.2
Let N' be a place refinement of N as in Definition 3.1.

(1) N and N' are transition-bisimilar.

(2) Subnet-contraction is a totally admissible operation.

(3) If $p' \neq p$ is implicit in N, then p' is implicit in N'. If additionally $l'(g_1) = l'(g_2) = \lambda$, N–p' is the subnet contraction of N'–p' as in Definition 3.1.

Proof. (1) Obviously, $\mathcal{B} = \{(M, M') \mid M'(\{p_1, p_2\}) = 1, M|_{-p} = M'|_{-p}, M(p) = M'(\{p_{in}, p_{out}\})\}$ is a transition bisimulation between N and N'. Observe that to match a firing $M[t\rangle M_1$ of N, it might be necessary to fire g_1 or g_2 in N' first if $p \in {}^\bullet t$.

(2) Clearly, subnet contraction decreases the termination function; no new structural λ/output conflicts are created since the same transitions are in structural conflict and only the transitions in $p_{out}^\bullet$ get new triggers (${}^\bullet p_{in}$), but since they have the dummy-transitions g_1 and g_2 as triggers in N', also 2.9(b) is preserved. (1) and Lemma 2.2 imply bisimilarity.

(3) The latter claim is obviously true. Regarding the first one, let $M' \in [M_{N'}\rangle$ and let M be the unique marking with $(M, M') \in \mathcal{B}$ from (1), and $M'|_{-p'}[t\rangle$. Since $p' \notin {}^\bullet g_1 \cup {}^\bullet g_2$, we have $M'[t\rangle$ for $t \in \{g_1, g_2\}$. Otherwise, ${}^\bullet t$ coincides in N and N' except that possibly p has to be exchanged with p_{out} if $t \in p^\bullet$ in N. Hence, $M|_{-p'}[t\rangle$ in N (since $(*)$ $M(p) \geq M'(p_{out}) \geq W'(p_{out}, t)$), $M[t\rangle$ by assumption and thus $M'[t\rangle$ due to bisimilarity. Observe that in this case, neither g_1 nor g_2 has to fire due to $(*)$ and $W'(p_{out}, t) = W(p, t)$. □

Proposition 3.3
Let N' be obtained from N by a gyroscope insertion with initial marking (in, out) as in Definition 3.1.

(1) N and N' are transition-bisimilar.

(2) If $p' \neq p$ is implicit in N, then p' is implicit in N'. N'–p' is the gyroscope insertion of N–p', in particular p is a new implicit place for N–p'.

(3) If the insertion is internal and for all $t' \in p_{out}^\bullet$ we have $l'(t') \in (Out \cup Int)^\pm$, then N is correct w.r.t. N' and vice versa.

Proof. (1) Let N'' be the result of the insertion of the new implicit place p, and let $\mathcal{B}_1$ be the transition-bisimulation between N and N'' as implied by Proposition 2.4. Proposition 3.2(1) implies that there is a transition-bisimulation $\mathcal{B}_2$ between N'' and N'. Transitivity of transition-bisimulations (cf. [KS07] for a slightly different version) implies that $\mathcal{B} = \mathcal{B}_1 \circ \mathcal{B}_2$ is a transition bisimulation between N and N' (for the common labelling l_T).

(2) The first claim follows from Propositions 2.5(1) and 3.2(3), the second from Propositions 2.5(2) and 3.2(3).

(3) Consider $\mathcal{B}$ from (1). One easily checks the condition of Definition 2.7: in general, a transition firing in N is matched by the same firing in N' and vice versa, except that firings of g_1 and g_2 in N' are ignored in N and a firing of a $t' \in p_{out}^\bullet$ might be preceeded by firings of g_1 and/or g_2. □

In particular, (3) is true for a gyroscope insertion with initial marking (in, out) between two transitions t and t', if these transitions are connected by an MG-path w with $M_N(w) = in + out$; in this case, the new place is a shortcut-place and hence implicit.

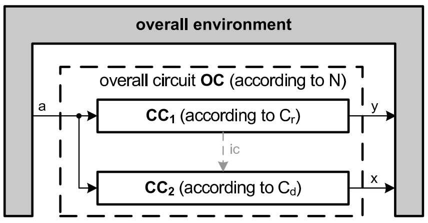

Figure 5: Block diagram of the desired system

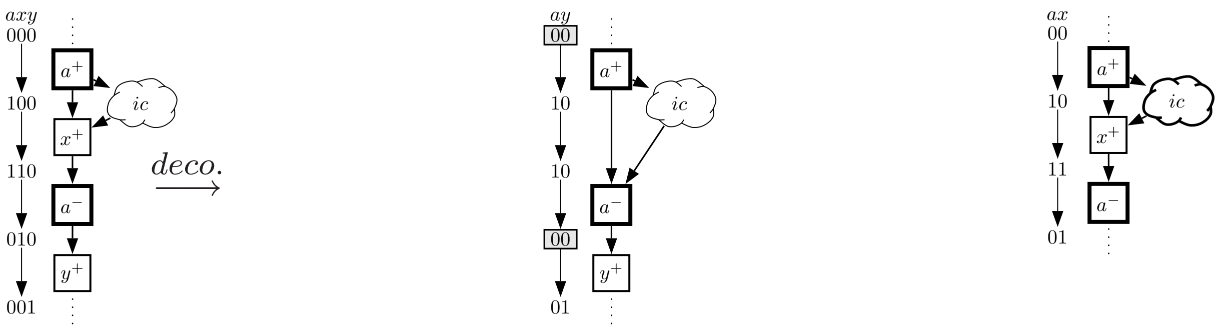

Figure 6: Avoidance of self-triggering by internal communication. *left*: specification (N) *middle*: crictical component (C_r) *right*: delay component (C_d)

4 Avoiding Self-Triggering

As mentioned above, the contraction based decomposition of [VW02, VK06] can lead to components having irreducible CSC conflicts: considering Figure 5, the desired overall circuit OC which is modelled by the overall STG N (see Figure 6 on the left) will be implemented using two smaller component circuits CC_1 for output y and CC_2 for output x that are acting in the same overall environment. CC_1 and CC_2 result from the logic synthesis of the component STGs C_r and C_d partially shown in the middle and on the right of Figure 6, accompanied by parts of their state graphs; the clouds should be ignored for now. C_r and C_d are the outcome of N's decomposition. As one can see, C_r has a CSC conflict between the highlighted states, since both have the same state vector $(a, y) = (0, 0)$ and in only one of them output y is enabled.

This conflict describes a so-called *(dynamic) self-trigger*, i.e. a firing sequence of two *input* transitions labelled with the same signal and complementary edges returning to the same state vector. The insertion of an *internal* signal transition ic^+ (at the position of the cloud) to solve this CSC-conflict is not possible, since it would not be input proper, cf. the discussion after Definition 2.7; hence, the conflict is irreducible.

In this context, we call the component having the self-trigger *critical*. Since such self-triggers are conflicts which arise in our benchmarks quite often we concentrate on them in this paper. To find them with a structural method, we define a *structural self-trigger* as two transitions t and t' which are labelled with complementary edges of the same input signal and satisfy $t \in {}^\bullet({}^\bullet t')$; a structural self-trigger is called *MG-self-trigger* if the place between t and t' is an MG-place. A structural self-trigger is necessary for a dynamic one.

Since there is a signal between a^+ and a^- in N, one can tune the decomposition not to contract the respective transition; instead, backtracking is performed and x is added as an input for C_r. Although this does help in some cases, in our experience an irreducible CSC conflict very often remains. Another possibility is to add output x to C_r; in this case, C_d will not be generated anymore and instead C_r is responsible for x and y and consequently larger. Note that this can be done easily only if the transition between a^+ and a^- that is contracted last is an output.

Input	specification N — critical component C_r with structural self-trigger t_2, t_4 delay component C_d with t_3 'between' t_2 and t_4 in N, $l(t_3) \in Out_d^{\pm}$
Output	modified specification N' — modified critical component C'_r — modified delay component C'_d
1	in N, perform an output gyroscope insertion for a fresh signal ic via an implicit place p and intermediate N'' such that in N'', $t_2 \in {}^\bullet p$, $t_3 \in p^\bullet$, $l(p^\bullet) \subseteq Out_d^{\pm}$ and $l({}^\bullet p) \subseteq In_r^{\pm}$; this gives N'
2	C'_r is obtained from reduction of N' with partition member $(In_r, Out_r + \{ic\})$
3	C'_d is obtained from reduction of N' with partition member $(In_d + \{ic\}, Out_d)$

Figure 7: Algorithm AVOID-0 for inserting internal communication between two components.

Hence, this method can only be used in some cases; and if it is applied repeatedly, it easily results in an uncontrollable growth of the remaining components, destroying the advantage of decomposition.

In this paper, we return to the idea to insert the internal signal ic between a^+ and a^-. The problem is that this requires the circuit environment to wait with lowering a for the occurrence of ic, although it does not even know about ic. The new idea is to achieve the desired causal relationship by making ic a communication signal between the components: we identify an output between a^+ and a^- in N (here x^+) and the respective so-called *delay component* C_d; then, we insert ic as an output in C_r and as an input in front of x^+ in C_d. This way, a^- waits for x^+, which in turn waits for ic. To view the situation from the point of C_r: its environment consists of all other components (including C_d) and the circuit environment, and this combined environment now indeed waits with lowering a for the occurrence of ic (i.e. C_d *delays* the overall environment such that a^- cannot arrive too fast causing malfunction of the circuit).

Finally, we hide ic (cf. set H in Definition 2.7) such that, from the point of view of N, we have inserted internal signal ic at the position of the cloud, and this is correct. This view is the key to prove the new method correct: first insert ic into N preserving correctness; since we can only decompose nets without internal signals, we regard the result as $N'/\{ic\}$ where ic is an output of N'. Second, we decompose N' with the same initial partition as before, except that ic is added to the outputs of C_r and the inputs of C_d. And indeed, in the reduction of C_r, x^+ is contracted putting ic into the right position.

Furthermore, it should be possible to avoid recomputation of the other components: for such a component, ic is lambdarised initially, so it should be possible to remove it immediately since it can even be removed when representing an internal signal. Then, reduction can proceed as in the case of N.

For an implementation of our idea, some problems remain to be solved. First, to ensure the practically important consistency, one cannot only insert ic^+ between a^+ and a^-, but one has to introduce a suitable ic^--transition somewhere else, too. This is problematic since it is unclear how to find a proper insertion point. Instead, the cloud is replaced by a gyroscope (simulating a toggle-transition). Another possibility is to replace it with a 4-phase handshake between the critical and the delay component; this approach will be investigated in future work.

Second, it is in no way obvious how to insert ic into a general N while preserving its behaviour, since the usual methods for event insertion would build the reachability graph of N, which we must avoid.[5]

The above considerations are formalised in the algorithm AVOID-0 in Figure 7. If there

[5]Unfolding based methods are more competitive but also not suited for very large specifications. Note that deciding whether a place is implicit is ExpSpace-hard.

are several components with irreducible CSC-conflicts, step 1 can be executed for each of them with respect to the modified specification, and afterwards every affected component is newly generated in a single second reduction pass. The following theorem and its proof can be easily adjusted for this scenario.

Proposition 4.1
The algorithm AVOID-0 is correct: if $(C_i)_{i\in I}$ is correct w.r.t. N when hiding H then $((C_i)_{i\in I'}, C'_r, C'_d)$ with $I' = I \setminus \{r, d\}$ is correct w.r.t. N' when hiding $H + \{ic\}$.

Proof. Proposition 3.3(3) implies that $N'/\{ic\}$ is correct w.r.t. N. In the following, we will show that the components $((C_i)_{i\in I'}, C'_r, C'_d)$ can be constructed from N' by the decomposition algorithm for some feasible initial partition. Then, $((C_i)_{i\in I'}, C'_r, C'_d)$ is a correct decomposition of N' and due to Theorem 2.8(2) a correct decomposition of N when hiding $H + \{ic\}$.

Let the gyroscope be defined as in Definition 3.1. First, the new partition is the original one exchanging the partition members for r and d as in Figure 7. This is feasible since

- the new output transitions g_1 and g_2 are in structural conflict only with each other and $s \in Out'_r$;
- g_1 and g_2 are only triggered by transitions with label in In'_r; they only trigger transitions with label in Out'_d (and each other) and $s \in In'_d$.

Now, consider a component C_j with $j \in I'$ and the corresponding initial component N'_j derived from N'. In N'_j, ic is lambdarised, which is feasible since ic only triggers outputs of C_d and C_r; hence $l'_j(g_1) = l'_j(g_2) = \lambda$. Then, Proposition 3.2 allows to apply subnet contraction to the inserted gyroscope yielding the STG N''_j. Clearly, in N''_j, the resulting place is implicit by definition of gyroscope insertion and can be deleted resulting in the STG N_j, which can be reduced to the final component C_j with the same sequence of operations applied to N_j to derived from N. □

Observe that the resulting decomposition is correct in the sense of Definition 2.7, but it is not even guaranteed that the structural self-trigger actually disappears. Also, the algorithm has to recalculate the affected components; instead we would like to apply gyroscope insertions directly on the calculated components. The next section presents an algorithmic solution for a (practically important) special case where this is possible.

5 Avoiding Recalculation

In this section, we will consider a very simple case of self-triggering where the specification is essentially an MG-path in the relevant part. We assume we are given a deterministic and live net N; also the latter is a common and sensible assumption on STGs, and in fact the proofs below only presuppose 2-liveness.

Let $(C)_{i\in I}$ be correct w.r.t. N when hiding H. In the critical component C_r, let there be an MG-self-trigger due to two input transitions t_2 and t_4 labelled with a^+ and a^- for $a \in In_N$. These transitions, the specification N and all components are inputs to the algorithm AVOID-1 in Figure 8, which returns with a modified specification and a modified critical and delay component; the critical component is always generated without a second decomposition pass, the delay component if possible.

In general, the algorithm returns FAIL whenever the structural conditions in line 1 for the specification are not fulfilled. A path between t_2 and t_4 (line 1) is guaranteed to exist since such a path exists in C_r and the reduction does not generate new paths, and usually (but not always) there will be an MG-path. However, the path might fail to be non-forking or there might be no

```
Input   specification $N$ — all components $(C_i)_{i \in I}$ — index $r$ of critical
        component
        MG-self-trigger place $p_s$ in $C_r$ with ${}^\bullet p_s = \{t_2\}$ and $p_s^\bullet = \{t_4\}$
Output  modified specification $N'$ with new output $ic$
        modified critical component $C'_r$ — modified delay component
        $C'_d$

1  in $N$, find an MG-path $w$ from $t_2$ to $t_4$ with the following
   properties
       return FAIL if no such $w$ exists
   $M_N(w) = M_{C_r}(p_s)$ and $l(w \cap T - \{t_2, t_4\}) \cap Sig_r^\pm = \emptyset$
   a transition $t_3$ of $w$ is labelled with an output edge $x^\pm$
   $w_2$, $w_3$ are the subpaths of $w$ from $t_2$ to $t_3$, from $t_3$ to $t_4$ resp.
   no transition of $w - \{t_4\}$ has more than one outgoing arc
2  choose $d$ such that $x \in Out_d$, choose a fresh signal $ic \notin Sig_N$
3  in $N$, perform an output gyroscope insertion between $t_2$ and
   $t_3$ for $ic$ with the initial marking $(M_N(w_2), 0)$, giving $N'$
4  in $C_r$, perform an output gyroscope insertion between $t_2$ and
   $t_4$ for $ic$ with the initial marking $(M_N(w_2), M_N(w_3))$, delete
   $p_s$; this gives $C'_r$
5  in $N$, find an MG-path $w_1$ from some transition $t_1$ to $t_2$ with
   the following properties ($t_1 = t_2$ is possible):
   $l(t_1) \in Sig_d^\pm$ and $l(w_1 \cap T - \{t_1\}) \cap Sig_d^\pm = \emptyset$
   no transition of $w_1 - \{t_1\} \cup w_2$ has more than one incoming arc
   if no such $w_1$ exists, return with $C'_d$ obtained from $N'$ and the
   respective partition member $(In_d + \{ic\}, Out_d)$ by reduction
6  in $C_d$, perform an input gyroscope insertion between $t_1$ and $t_3$
   for $ic$ with the initial marking $(M_N(w_1 \cup w_2), 0)$, giving $C'_d$
```

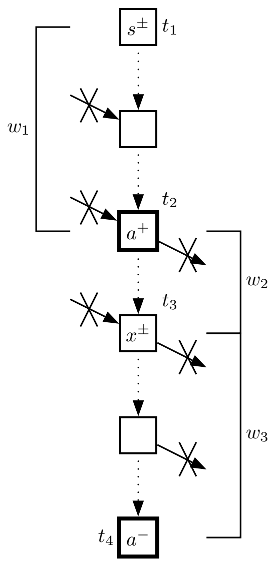

Figure 8: Algorithm AVOID-1 for inserting internal communication without recalculation of the components.

proper output which can help to destroy the self-triggering. In this case, the self-triggering might be avoided by adding additional inputs to C_r as discussed in the previous section. Otherwise, a possible output x and the delay component C_d producing it are chosen. In the rest of the algorithm, the corresponding gyroscope insertions into N and the components C_r and C_d take place where for C_d a proper starting transition t_1 has to be found (line 5). If the latter is not possible, C'_d can be recalculated from the modified specification N'.

A concrete implementation has some freedom since there might be more than one suitable ouput on w. In such a case, one might choose the output x and the corresponding delay component C_d such that a path w_1 exists at all or is better suited than others. If $a \in Sig_d$, we have $t_1 = t_2$ and the conditions for w_1 are trivially fulfilled. In this case, t_2 might additionally be a syntactical trigger of t_3 in C_d; then, it is possible that the signal a is no longer necessary and might lambdarised in C'_d and contracted, resulting in a correct decomposition as implied by Theorem 2.8(1).

Theorem 5.1
The algorithm AVOID-1 is correct in the sense of Proposition 4.1. The self-trigger in C_r is not present in C'_r anymore.

Proof. Regarding termination, observe that N is finite and therefore the path searching terminates. In the rest of the proof, we show that C'_r and C'_d can be derived via a correct reduction from N', and the claim then follows from Proposition 4.1.

Consider the component C'_r. Let $(R_m)_{0 \le m \le n}$ be the intermediate STGs encountered during the generation of C_r. We will now show that the same sequence of reduction operations can be applied to the initial component N'_r, resulting in a sequence $(R'_m)_{0 \le m \le n}$ of intermediate STGs with $R'_n = C'_r$ and the following properties: in every R'_m

(1) there is exactly one transition $t^m \in p_{out}^\bullet$ with $t^m \neq t_2$.

(2) there are non-forking paths v_1^m from t_2 to t^m and v_2^m from t^m to t_4 (t_4 can have more than one place in its postset)

(3) R'_m can be derived from R_m by an input gyroscope insertion between t_2 and t^m with initial marking $(M_N(w_2), M_N(w_3) - M_{R'_m}(v_2^m))$

Clearly, this is initially true: for R_0 and R'_0 choose $t^0 = t_3$, $v_1^0 = w_2$ and $v_2^0 = w_3$. So let the claim be fulfilled for some R_m and R'_m and let R_{m+1} be obtained from R_m by some reduction operation.

Deletion of an implicit place p' Observe that no place q of v_1^m or v_2^m is implicit since Lemma 2.6 implies in this case that there is another path between ${}^\bullet q$ and $q^\bullet$, a contradiction to v_i^m being non-forking. Then, Proposition 3.3(2) implies that N'–p' is the gyroscope insertion of N–p'. Choose $v_i^{m+1} = v_i^m$ $(i = 1, 2)$ and $t^{m+1} = t^m$.

Deletion of a redundant transition t Observe that a transition on $v_1^m\,v_2^m$ is adjacent to an MG-place on $v_1^m\,v_2^m$ that is also adjacent to another transition, since $v_1^m\,v_2^m$ leads from t_2 to $t_4 \neq t_2$; hence these transitions are not redundant. Also, t is not adjacent to p, and Proposition 2.5(4) then implies that t is also redundant in R'_m, and R'_{m+1} is obtained from the gyroscope insertion in R_{m+1} due to Proposition 2.5(3).

Contraction of a transition t Confer Figure 9.

(1) $t \notin v_1^m, v_2^m$: claim obviously fulfilled, because in the intermediate R''_m and hence also in R''_{m+1}, p is a shortcut place w.r.t v_1^m. In all other cases, the path $v_1^m\,v_2^m$ is modified. Here, it is important that all transitions on $v_1^m\,v_2^m$ (except t_4) are non-forking; this property also holds for the modified path.

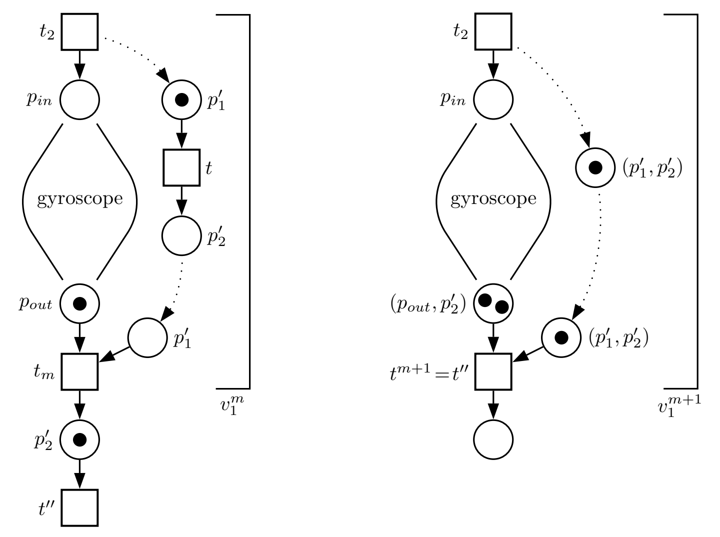

Figure 9: For the proof of Theorem 5.1. Cases (2) and (3) of transition contraction.

(2) $t = t^m$: by assumption, there is only one place p'_2 in the postset of t^m ($t = t_4$ is not possible since t_4 is not λ-labelled). The contraction of t^m is possible in R'_m since the additional place $p_{out} \in {}^\bullet t_m$ does not render the contraction insecure. Let $v_1^m = t_2 v p'_1 t^m$ for some path v and $v_2^m = t^m p'_2 v'$ for some path v'. The places p'_1, p'_2 and p_{out} are replaced in R'_{m+1} by the places (p'_1, p'_2), (p_{out}, p'_2); for simplicity we identify the latter with p_{out}. Then, t_{m+1} is the single transition in $p_2'^\bullet$ and $v_1^{m+1} = t_2 v (p'_1, p'_2) t_{m+1}$, $v_2^{m+1} = v'$ and furthermore:

$$\begin{array}{rcl} & & M_{R'_{m+1}}((p_{out}, p'_2)) \\ \text{(definition)} & = & M_{R'_m}(p_{out}) + M_{R'_m}(p'_2) \\ \text{(induction)} & = & M_N(w_3) - M_{R'_m}(v_2^m) + M_{R'_m}(p'_2) \\ & = & M_N(w_3) - (M_{R'_m}(t^m p'_2 v') - M_{R'_m}(p'_2)) \\ & = & M_N(w_3) - M_{R'_m}(v') \\ & = & M_N(w_3) - M_{R'_m}(v_2^{m+1}) \end{array}$$

p_{in} is not touched by the contraction and therefore (3) follows. Observe that the initial marking of $\{p_{in}, p_{out}\}$ increases by $M_{R'_m}(p'_2)$ and so does the marking of v_1^{m+1} compared to that of v_1^m; hence, the gyroscope insertion leading to R'_{m+1} is indeed based on a shortcut place p.

(3) $t \in v_1^m - \{t^m\}$: let $v_1^m = t_2 v' p'_1 t p'_2 v'' t^m$ for some paths v' and v''. Since t is not adjacent to the inserted gyroscope, the contraction is also possible in the intermediate R''_m leaving p being a shortcut place, and also in R'_m.

In R'_{m+1}, choose $v_1^{m+1} = t_2 v' (p_1, p_2) v'' t^m$, $v_2^{m+1} = v_2^m$ and $t^{m+1} = t^m$.

(4) $t \in v_2^m - \{t^m\}$: similar to (3)

In R'_n, $t^n = t_4$, $v_1^n = t_2 p_s t_4$ and $v_2^n = t_4$ since all transitions of $w_3 - \{t_4\}$ have been contracted. Hence, $M_{R'_n}(p_{in}) = M_N(w_2)$ and $M_{R'_n}(p_{out}) = M_N(w_3)$. The gyroscope insertion first inserts a place p resulting in an intermediate STG R''_n such that $M_{R''_n}(p) = M_{R'_n}(p_{in}) + M_{R'_n}(p_{out}) = M_N(w_2 \cup w_3) = M_N(w) = M_N(p_s)$. Therefore, p_s is implicit in R''_n and due to Proposition 3.2 also implicit in R'_n. Deleting it there results in the final component C'_r returned by the algorithm.

If C'_d is recalculated from N', it is obviously correct. Otherwise, the proof is analogous to the previous one. Observe that this time p_{out} stays untouched while p_{in} is combined with places from w_1; in particular, w_1 has to be non-joining to avoid the 'duplication' of p_{in}. □

The theorem above shows that we have inserted into the self-trigger an output that records the occurrence of a^+. This removes the original irreducible CSC conflict, but it could very well be the case that there still is some CSC conflict: the first a^+ is recorded by ic^+, the second one by ic^-; so the state vector before the first a^+ could easily be the same as the one after the second a^-. What we achieved is that this CSC conflict can now be solved by inserting internal signals into C_r alone as described in [CKK+02] and proven correct in the sense of Definition 2.7 in [SV07].

In fact, the irreducible CSC conflict can also vanish completely if the two state vectors just mentioned differ in other signals; see [WW07a] for a small example. In the experiments reported in the next section, this is actually always the case; here, the reason is that always at least two gyroscopes are inserted 'non-concurrently'.

6 Experimental Results

The approach of the previous section was implemented as part of our existing decomposition tool DesiJ [Sch07]. In this section we compare it with standard logic synthesis by Petrify [CKK+02] and Mpsat [KKY04] and syntax-directed translations of handshake circuits.

In the latter, a circuit behaviour is specified e.g. by a Balsa program [EB02]; following its syntax, this is translated into a 'netlist' of predefined handshake components (HCs), which have (handmade) standard implementations. To build the final circuit, these are connected with handshake channels (usually using the 4-phase protocol) as specified by the netlist. This approach is very fast and results in robust (i.e. delay insensitive) implementations of very large specifications. However, due to the extensive use of handshake communication, the resulting circuits are heavily overencoded and therefore large and slow.

We aim at *resynthesis* of such HC netlists in order to remove unnecessary handshake signals as follows: each HC is specified by a (usually small) STG, and these are composed in parallel according to the netlist to get one overall STG. In this STG, we lambdarise and contract the intercomponent handshake signals such that only the external behaviour remains. The result is then used as input specification for our approach.

The SeqParTree (SPT) benchmarks examples used here were first used in [CC06] and consist of two types of handshake components with one passive and two active ports each. A *sequencer* (;) waits for a handshake on its passive port and performs one handshake each on its both active ports in sequence, while a *paralleliser* (||) performs them in parallel. The SPT benchmarks are trees of sequencer and paralleliser components such that the root is a sequencer which has two parallelisers as children and each paralleliser can have two sequencers as children again etc. SPT-n is such a tree with n levels (consisting of $2^n - 1$ components); Fig. 11 shows SPT-3.

All benchmarks were performed on a Pentium 4 HT with 3 GHz and 2 GB RAM. The DesiJ part of the table in Figure 10 reports the results of the decomposition based synthesis of SPT-2 to SPT-7 by applying the new approach using Petrify for logic synthesis of the components. The columns report the number of places, transitions and input/output signals of the corresponding specification. CPU times are given in seconds for the decomposition (*deco*), the insertion of internal communication (*int*) and the synthesis of the components with Petrify (*syn*). The *sig* column reports the number of added internal communication signals, and in column *lit*, the number of literals for a complex gate implementation is shown. This applies also for the Petrify, Mpsat and Balsa parts; note that for the first two, *int* denotes the number of internal signals which are introduced during synthesis.

In our examples, the new approach was able to turn components with irreducible CSC-conflicts directly to components with CSC, i.e. no additional internal signals had to be introduced during synthesis; cf. the discussion in the previous section.

	DesiJ (+ Petrify)					Petrify			Mpsat			HDL
SPT	$\|P\|$ / $\|T\|$	$\|In\|$ / $\|Out\|$	CPU (deco/int/syn/Σ)	*sig*	*lit*	*sig*	CPU	*lit*	*sig*	CPU	*lit*	*lit*
2	23 / 20	5 / 5	0 / 0 / 1 / 1	8	29	4	10	14	4	0	37	54
3	39 / 36	9 / 9	0 / 0 / 1 / 1	12	47	5	59	36	7	26	59	86
4	91 / 68	17 / 17	0 / 0 / 8 / 8	40	123	9	9631	70	11	741	134	270
5	155 / 132	33 / 33	1 / 0 / 9 / 10	56	195	mem. overflow			mem. overflow			398
6	403 / 260	65 / 65	8 / 12 / 1297 / 1317	208	579	mem. overflow			mem. overflow			1134
7	659 / 516	129 / 129	29 / 27 / 1372 / 1428	272	867	mem. overflow			mem. overflow			1646

Figure 10: Results of new approach compared to pure logic synthesis and syntax directed translation. The CPU times are given in seconds.

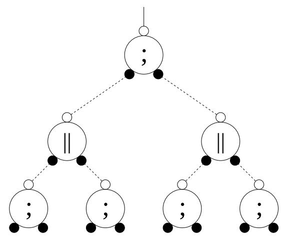

Figure 11: SPT-03. Black dots are active handshake ports, white ones are passive. They are connected by handshake channels consisting of two signals, one output per port. The dotted channels are lambdarised in the specification.

To the best of our knowledge, none of the existing STG synthesis techniques is able to synthesise such large specifications. With the method of [CC06], resynthesis of SPT-5 takes 132 seconds and yields an implementation with 269 literals. In contrast, we are able to synthesise SPT-5 in 10 seconds yielding a much smaller implementation with 195 literals. Moreover, our new approach can even synthesise SPT-6 and SPT-7, in less than 25 minutes. Observe that e.g. SPT-7 is not much harder to resynthesise than SPT-6: intuitively, parallelisers add concurrent behaviour to a specification, increasing the state space drastically; since SPT-7 is derived from SPT-6 by adding only additional sequencers, this effect does not occur.

The new approach is also successful regarding the goal of resynthesis, i.e. the size of the resulting circuit was decreased when compared to the direct translation. As one can see, each implementation resulting from a Balsa translation leads to larger circuits (in terms of literals[6]) than the synthesis according to our approach. On average, we reached a reduction of 50%. The impact on the circuit's performance will be investigated in future work.

Compared to pure logic synthesis with Petrify or Mpsat our circuit area is slightly better than that of Mpsat, but twice as large as that of Petrify. However, we are not only able to synthesise circuits by orders of magnitude faster, but we can even synthesise SPTs with more than 4 levels, which is impossible with pure logic synthesis due to memory overflows.

7 Conclusions and Outlook

Summing up, irreducible CSC-conflicts resulting from STG decomposition as in [VW02, VK06] can often be solved by introducing internal communication between the components. Since this approach is purely structural, we consider it as a breakthrough in decomposition based logic synthesis. In particular, the resynthesis of handshake specifications of a size never reached before is now possible.

Furthermore, this method also contributes to the CSC solving problem of large specifications such that CSC is not solved for the overall STG (as in [CC06]), but it is broken down to the CSC solving of the smaller components which is a well-known and optimised technique.

There are several open problems which will be investigated in the future: a gyroscope insertion is easy but potentially doubles the state space of the respective component (since the state of the new signal is independent of all other signals), which can prevent synthesis. Alternatives are:

- the insertion of a two-way communication according to a 4-phase handshake protocol: the critical component sends ic^{+}_{req} to the delay component, which answers with ic^{+}_{ack} and then

[6]These costs are calculated by summing up the costs of each handshake component. According to [vB93], the costs of a sequencer and a paralleliser are 8 and 23 literals resp.

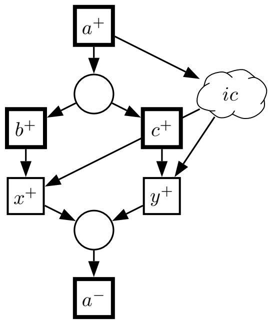

Figure 12: Specififcation with branching and merging resulting in two delay components C_x and C_y.

ic^-_{req} and ic^-_{ack} follow. This doubles the number of inserted signals but could keep the state space small, and thus might be needed for the synthesis of larger specifications.[7].

- the insertion of a single signal edge ic^+, and the insertion of ic^- somewhere else such that consistency is preserved.[8] In particular, if two gyroscopes are inserted into one component as in our examples, one could be replaced by ic^+ and the other by ic^-. However, this might require a state space analysis of the component, since the respective signa edges must not be enabled concurrently to guarantee consistency. Hence, this might only be suitable for small components.

Furthermore, detection of irreducible CSC conflicts has to be improved, i.e. we will consider more general (i.e. longer) input sequences between two conflicting markings than self-triggers; such sequences often have the form $a^+b^+a^-b^-$. There can also be irreducible CSC-conflicts without an input sequence leading from one marking to the other; e.g. the two markings could be reached from the same marking with a^+b^+, b^+a^+ resp.

Also, we have to deal with MG-self-triggers that do not arise from an MG-path of the specification, which will involve more than one delay component. For the STG in Figure 12, there might be an MG-self-trigger for input a, which can be resolved with the depicted internal communication resulting in two delay components C_x and C_y.

[7]Probably ic_{ack} can be implemented very simple.

[8]This is quite similar to ordinary CSC solving, except that the constraints differ somehow. Hence, the same techniques used there might be helpful.

References

[And83] C. André. Structural transformations giving B-equivalent PT-nets. In Pagnoni and Rozenberg, editors, *Applications and Theory of Petri Nets*, Informatik-Fachber. 66, 14–28. Springer, 1983.

[BE00] A. Bardsley and D. A. Edwards. The Balsa asynchronous circuit synthesis system. In *FDL2000: Forum on Design Languages. European Electronic Chips and Systems Design Initiative (ECSI)*, 2000.

[Ber87] G. Berthelot. Transformations and decompositions of nets. In W. Brauer et al., editors, *Petri Nets: Central Models and Their Properties*, Lect. Notes Comp. Sci. 254, 359–376. Springer, 1987.

[CC03] J. Carmona and J. Cortadella. ILP models for the synthesis of asynchronous control circuits. In *Proc. IEEE/ACM Int. Conf. on Computer Aided Design*, pages 818–825, 2003.

[CC06] J. Carmona and J. Cortadella. State Encoding of Large Asynchronous Controllers. In *Proc. DAC'06*, pages 939–944. IEEE, 2006.

[Chu87] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications.* PhD thesis, MIT, 1987.

[CKK$^+$02] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces.* Springer, 2002.

[EB02] D. Edwards and A. Bardsley. Balsa: an Asynchronous Hardware Synthesis Language. *The Computer Journal*, 45(1):12–18, 2002.

[Ebe92] J. Ebergen. Arbiters: an exercise in specifying and decomposing asynchronously communicating components. *Sci. of Computer Programming*, 18:223–245, 1992.

[KKY04] V. Khomenko, M. Koutny, and A. Yakovlev. Logic synthesis for asynchronous circuits based on Petri net unfoldings and incremental sat. In Kishinevsky M. and Ph. Darondeau, editors, *ACSD 2004*, pages 16–25. IEEE, 2004.

[KNE$^+$05] N. Karaki, T. Nanmoto, H. Ebihara, S. Utsunomiya, S. Inoue, and T. Shimoda. A flexible 8b asynchronous microprocessor based on low-temperature poly-silicon TFT technology. In *ISSCC '05: Proceedings of the Solid-State Circuits Conference, 2005*, pages 272–598. Seiko Epson Corp., Nagano, Japan, 2005.

[KS07] V. Khomenko and M. Schaefer. Combining decomposition and unfolding for STG synthesis. In *ATPN '07: Applications and Theory of Petri Nets and Other Models of Concurrency*, Lect. Notes Comp. Sci. 4024, pages Springer, 223–243, 2007.

[Rei85] W. Reisig. *Petri Nets.* EATCS Monographs on Theoretical Computer Science 4. Springer, 1985.

[Sch07] M. Schaefer. DesiJ - a tool for decomposition. Technical Report 2007-11, University of Augsburg, http://www.informatik.uni-augsburg.de/de/forschung/reports/, 2007.

[SV07] M. Schaefer and W. Vogler. Component refinement and CSC solving for STG decomposition. *Theoret. Comput. Sci.*, 388(1–3):243–266, 2007.

[SVJ05] M. Schaefer, W. Vogler, and P. Jančar. Determinate STG decomposition of marked graphs. In G. Ciardo and P. Darondeau, editors, *ATPN 05*, Lect. Notes Comp. Sci. 3536, 365–384. Springer, 2005.

[SY05] D. Sokolov and A. Yakovlev. Clockless circuits and system synthesis. In *IEE Proceedings – Computers Digital Techniques*, volume 152, pages 298 – 316, 2005.

[vB93] Kees van Berkel. *Handshake circuits: an asynchronous architecture for VLSI programming*. Cambridge University Press, New York, NY, USA, 1993.

[vBJN99] C. H. Kees van Berkel, Mark B. Josephs, and Steven M. Nowick. Scanning the technology: Applications of asynchronous circuits. In *Proceedings of the IEEE*, volume 87, pages 223–233, 1999.

[vGvBP$^+$98] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80c51 microcontroller. In *ASYNC '98*, pages 96–107. IEEE, 1998.

[VK06] W. Vogler and B. Kangsah. Improved decomposition of signal transition graphs. *Fundamenta Informaticae*, 76:161–197, 2006.

[VW02] W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. In J. Cortadella et al., editors, *Concurrency and Hardware Design*, Lect. Notes Comp. Sci. 2549, 152 – 190. Springer, 2002.

[WB00] R. Wollowski and J. Beister. Comprehensive causal specification of asynchronous controller and arbiter behaviour. In A. Yakovlev, L. Gomes, and L. Lavagno, editors, *Hardware Design and Petri Nets*, pages 3–32. Kluwer Academic Publishers, 2000.

[WW07a] D. Wist and R. Wollowski. Avoiding irreducible CSC conflicts in component STGs. In *Proceedings of the 19th UK Asynchronous Forum*. Imperial College London, 2007.

[WW07b] D. Wist and R. Wollowski. STG decomposition: Avoiding irreducible CSC conflicts by internal communication. Technical Report 20, HPI, University of Potsdam, 2007. ISBN 978-3-940793-02-7, ISSN 1613-5652.

[YOM04] T. Yoneda, H. Onda, and C. Myers. Synthesis of speed independent circuits based on decomposition. In *ASYNC 2004*, pages 135–145. IEEE, 2004.