Fundamenta Informaticae 108 (2011) 1–21
DOI 10.3233/FI-2011-300
IOS Press

A Scheduling Strategy for Synchronous Elastic Designs

Josep Carmona[*]
Universitat Politècnica de Catalunya
Barcelona, Spain
jcarmona@lsi.upc.edu

Jorge Júlvez
Universidad de Zaragoza
Zaragoza, Spain
julvez@unizar.es

Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain
jordi.cortadella@upc.edu

Michael Kishinevsky
Intel Corporation
Hillsboro, USA
michael.kishinevsky@intel.com

Abstract. With the scaling of process technologies, communication delays represent a bottleneck for the performance of circuits. One of the main issues that has to be handled is the variability of such delays. Latency-insensitive circuits offer a form of elasticity that tolerates variations in those delays. This flexibility usually requires the addition of a control layer that synchronizes the flow of information. This paper proposes a method for eliminating the complexity of the control layer, replacing it by a set of iterative schedulers that decide when to activate computations. Unlike previous approaches, this can be achieved with low complexity algorithms and without extra circuitry.

1. Introduction

Latency insensitive (or *synchronous* elastic) systems have been suggested by a few research groups as a form of discretized asynchronous systems (see, e.g., [5, 14, 11, 7]). Such systems are elastic in the sense that they can tolerate dynamic and static changes in *latencies* of computation and communication components as counted in the number of clock cycles.

Consider Fig. 1(a), depicting a system with four functional units A,B,C,D, each one delivering the result to a register (shadowed boxes), and assume that every functional unit has a 1-cycle delay. If for scalability reasons, when migrating to a new technology, the units connected through the long wires $C \rightarrow A$ and $D \rightarrow B$ cannot communicate within one clock cycle, intermediate registers must

[*]Address for correspondence: Universitat Politècnica de Catalunya, Carrer de Jordi Girona, Barcelona, Spain

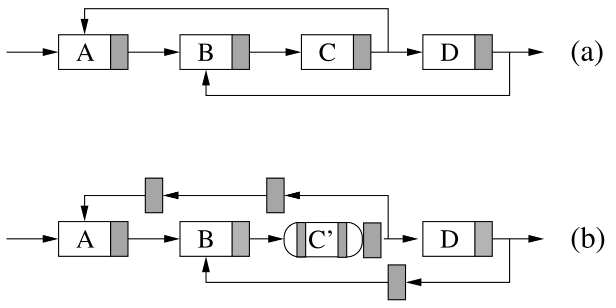

Figure 1. (a) A system with four functional units, (b) insertion of intermediate registers and incorporation of a variable-latency unit.

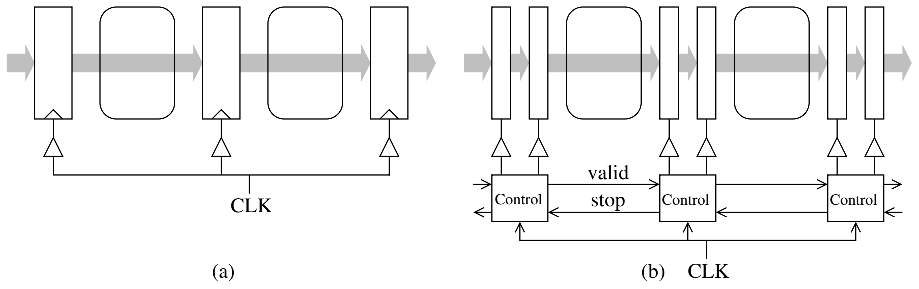

Figure 2. (a) A synchronous circuit: round boxes denote combinational logic and square boxes denote registers, (b) Elastic circuit.

be inserted (as shown in part (b)), to allow the communication (this transformation is called *pipelining*). Other typical situation is to incorporate variable latency units like C in Fig. 1 (b), that takes two cycles for long operands and one cycle otherwise. In synchronous circuits, this type of modifications require drastic manual changes to accomodate the new delays of the system. In contrast, these problems are avoided if elasticity is adopted, due to the aforementioned capability of tolerating communication and computation delays.

The implementation of elastic systems relies on synchronous handshakes containing stalling signals, optimized for their use in coordination with the clock. In Figure 2 one can see the main differences between a synchronous circuit (Figure 2(a)), and its elastic counterpart (Figure 2(b)): a control network is added to coordinate the latching of data in the registers, using the valid/stop signals that comunicate through adjacent stages. Moreover, to incorporate elasticity, an elastic FIFO of capacity two and latency one is added on each stage. This FIFO can be efficiently implemented with two latches (as shown in Figure 2(b)) using a slightly modified master-slave latch structure with the area and power cost similar to the cost of the standard synchronous register. As in a conventional synchronous circuit, the clock signal (CLK in the figure) is used to establish the time instants when data is available. Full details of elastic circuits can be found in [7].

Nevertheless elasticity can have area and routing overhead, given that additional handshake wires and registers are inserted into the circuit area. Scheduling latency-insensitive designs [2, 3, 9] is a circuit-level transformation that replaces the handshake controllers in each combinational block by iterative

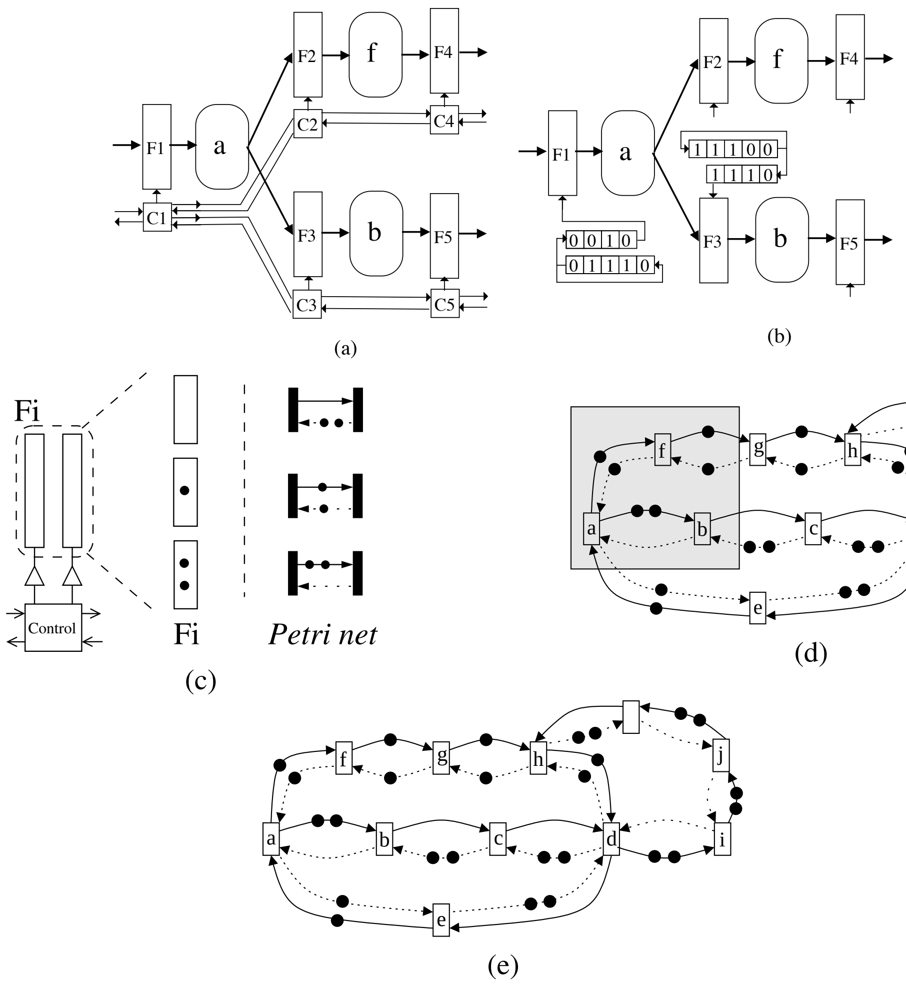

Figure 3. (a) Fragment of an elastic system (Ci are controllers, Fi are latches, a, b and f denote combinational logic), (b) schedule replacing the control layer (only shown the registers for a and b), (c) configurations of a latch and the corresponding Petri net modeling, (d) Initial Petri net model extended with stall information for the complete system (grey box represents the part depicted in (a)-(b)), (e) transformation to acquire optimal performance.

schedulers (e.g. shift registers with the activation pattern for each block), thus eliminating the need for the handshake signals. Fig. 3(a)-(b) shows an example of such transformation, where all the controllers Ci are replaced by shift registers that activate the latching of data each time a one is produced on their output. The idea is that once the critical cycle of the latency-insensitive system is known, the system is transformed to work at the speed of that cycle, and fast branches are delayed to avoid the need of stall signals. When the routing constraints are hard to satisfy or when area reduction is a goal, static scheduling might be an elegant alternative.

Some attempts for applying scheduling to latency-insensitive circuits have been proposed in the literature. In [9], a scheduling technique is presented for simplifying latency insensitive designs, based on a previous technique for scheduling using software pipelining techniques [3]. Another approach is presented in [2]. Both approaches are based on the following idea: the system must be *equalized* to avoid the need for stalling signals, i.e. *every elementary cycle must work at the same throughput*. When this cannot be achieved, different solutions are presented, in space or time: in [9], the clock is divided into multiphases to ensure the equalization (time), while [2] introduces *fractional registers* to be inserted in fast branches when the equalization of such branches need a fractional delay (space). Implicitly, both techniques assume a predefined mode of operation (called *ASAP*, i.e. As Soon As Possible, in [2]) that requires the activation of a computation node as soon as the inputs are available.

1.1. Example and contributions

Let us use the example of Fig. 3 to illustrate the contributions of this paper. An elastic circuit is shown in Fig. 3(a), where latches of each stage are grouped together into one elastic FIFO of capacity two F_i, as shown in Fig. 3(c). Fig. 3(c) also shows how to formally model the different configurations of a stage by means of the marking of a pair of complementary arcs of a Petri net: when the latch is empty, the dashed arc of the pair holds the two tokens, when it is half-full (denoted by one dot inside the latch), one token is assigned to each arc, and when the latch is full two tokens reside in the solid arc (see Sect. 2.3 for a formal definition of the model). The initial model, shown as a Petri net in Fig. 3(d), describes the behavior of the system where transitions (boxes) represent combinational logic in stages, complementary arcs denote latches and tokens in arcs represent the initial data distribution, as explained in Fig. 3(c). The model explicitly contains stalling information in the *backward arcs*, denoted by dotted lines. Stalling arises in a stage when the corresponding backward arc is unmarked.

The main contributions of this paper are the following[1]:

1) *Enhance the schedules by incorporating stalling information into the model.* Previous works require the equalization of every elementary cycle of the system. This might be too restrictive (both in complexity of the methods and in the feasibility of the solution to satisfy), and in this way we found that if the ASAP mode of activation is dropped, the constraint of equalization of every cycle can be left out. The core idea in our approach is that stall information is explicitly inserted in the initial model of the latency-insensitive circuit, and therefore the behavior of the model contains this information and the consequent analysis and schedule computation takes into account the stalling. With this optimization, the throughput of the system will still be governed by the most stringent cycles but not every cycle must be working at the slowest speed.

In circuits where the maximal throughput cannot be achieved due to stalls, performance optimization techniques are applied before computing the schedules. This is illustrated in the example of Fig. 3. The model in Fig. 3(d) reveals an important information: the backward cycle $\{d, h, j, i\}$ determines the performance of the system, since its ratio tokens/delay is the minimal ($1/4$). Fig. 3(e) shows a possible transformation (called *recycling* [6]) to avoid the throughput degradation due to the stalling policy of an elastic system. The transformed system has throughput $3/5$. Finally, the schedules are found by simulating the system represented in Fig. 3(e). The implementation of these schedules (e.g. using shift registers) replaces controllers and the wires corresponding to handshake signals, as shown in Fig. 3(b).

2) *An efficient method to compute a partition of the set of computation blocks* is provided, for which any block in a given partition can share the same scheduling register. In this sense, a second level of optimization is provided in which the set of registers needed for the scheduling might be significantly

[1]This is an extended version of [8].

reduced. In the example of Fig. 3, our technique will find that the registers for b, f and i can share the same iteration scheduler, and hence only one instead of three is needed. A theoretical result of the paper is that the number of schedules sufficient for controlling the elastic system is upper bounded by the number of blocks in critical cycles. In an idealistic case, the schedules of all computation blocks can be mapped to a schedule from a block in a critical cycle. In practice, this might be infeasible due to physical constraints. We have implemented a layout-aware partition technique that takes this information into account.

The remainder of this paper is organized as follows: Section 2 provides the necessary theory for the paper. In Section 3, a method for finding the clusters of computation blocks that may share the iteration scheduler without performance degradation is presented (contribution 2) in Section 1.1. The scheduling technique of Section 4 is then combined with the clustering technique of Section 3, deriving a complete flow for scheduling elastic designs. Finally, experiments on the combination of scheduling and clustering are presented, illustrating how the complexity of an elastic design can be significantly reduced by the application of our approach.

2. Marked Graph Model

This section presents the class of timed marked graphs that is used for modeling elastic systems. Although the paper is self-contained the reader can be referred to [22] for a survey on Petri Nets.

2.1. Timed Marked Graphs

Definition 2.1. A *Timed Marked Graph* (TMG) is a tuple $G = (T, A, M_0, \delta)$, where:

- T is a set of transitions (also called nodes).
- $A \subseteq T \times T$ is a set of directed arcs.
- $M_0 : A \to \mathbb{N} \cup \{0\}$ is the initial marking. that assigns an initial number of tokens to each arc.
- $\delta : T \to \mathbb{R}$ assigns a non-negative delay to every transition.

Given a transition $t \in T$, ${}^\bullet t$ and $t^\bullet$ denote the set of incoming and outgoing arcs of t, respectively. Given an arc $a \in A$, ${}^\bullet a$ and $a^\bullet$ refer to the source and target transition of a respectively. A transition t is *enabled* at a marking M if $M(a) > 0$ for every $a \in {}^\bullet t$. An enabled transition t fires after $\delta(t)$ time units. The firing of t removes one token from each input arc of t, and adds one token to each output arc of t. This paper focuses on synchronous elastic circuits. Hence, all transitions have the same delay: $\delta(t_i) = \delta(t_j)$ for every $t_i, t_j \in T$. Sometimes we denote an arc $(t_i, t_j) \in A$ by simply $t_i t_j$.

2.2. State equation and token preservation

Some useful basics of strongly connected MGs are [22]:
Reachability. We say that the marking M is *reachable* from M_0 if there is a sequence of transitions that can fire starting from M_0 leading to M.

State equation. Let $\mathbb{C}$ be the $n \times m$ incidence matrix of the MG with rows corresponding to n arcs and columns to m transitions.

$$\mathbb{C}_{ij} = \begin{cases} -1 & \text{if } t_j \in a_i^\bullet \setminus {}^\bullet a_i \\ +1 & \text{if } t_j \in {}^\bullet a_i \setminus a_i^\bullet \\ 0 & \text{otherwise} \end{cases}$$

The marking M is reachable from the initial marking M_0 iff the state equation

$$M = M_0 + \mathbb{C} \cdot \sigma \geq \mathbf{0} \tag{1}$$

is satisfied for some firing count vector σ (the j's component of σ corresponds to the number of times transition t_j has fired). Notice that the equation (1) provides a necessary and sufficient condition for reachability of a marking for the class of nets considered in this paper [22].

Token preservation. If the marking M is reachable then

$$\sum_{a \in \mathbf{c}} M(a) = \sum_{a \in \mathbf{c}} M_0(a)$$

for every cycle $\mathbf{c}$ of the MG.

2.3. Elastic Marked Graphs

Definition 2.2. An *Elastic Marked Graph* (EMG) is a TMG such that for any arc $a \in A$ there exists a complementary arc $a' \in A$ satisfying the following condition ${}^\bullet a = a'^\bullet$ and ${}^\bullet a' = a^\bullet$.

A labelling function L maps all arcs of an EMG as forward or backward $L : A \rightarrow \{F, B\}$ such that $L(a) = F$ iff $L(a') = B$.

We assume that for any complementary a and a', $M(a) + M(a') = 2$ (see any pair of complementary arcs in Fig. 3(d)), because this is the minimal capacity needed to have elasticity [5][2]. Therefore, all EMGs in this paper are 2-bounded (an arc cannot have more than two tokens). Semantically, the pair $\{a, a'\}$ represents the state of a FIFO between two stages of the elastic system. Assume that $L(a) = F$ and $L(a') = B$. We say that the FIFO is full when $M(a) = 2$, $M(a') = 0$; when $M(a) = 0$, $M(a') = 2$ we say that there is a *bubble* in the FIFO. For instance, the FIFO represented by the arc pair $\{c, d\}$ in Figure 3(d) contains a bubble. $M(a)$ represents the number of information items inside the buffer, while $M(a')$ represents available free space in the state of the system that corresponds to the marking M. $M_0(a)$, and $M_0(a')$ represents the corresponding values at the time of system initialization after the reset.

Without loss of generality, we model elastic systems with strongly connected EMGs. For open systems interacting with an environment, it is possible to incorporate an abstraction of the environment into the model by a transition that connects the outputs with the inputs.

[2]Clearly, the more capacity in the channels the less degradation in the system throughput [21].

2.4. Throughput of an EMG

The performance of an EMG can be measured by the throughput of its transitions. Given that we are considering strongly connected EMGs, in the steady state all transitions have exactly the same throughput, Θ. The throughput intuitively corresponds to the number of times each operation is performed on average per unit of time during the infinitely long execution of the system.

If C is the set of simple directed cycles in an EMG, its throughput can be determined as [23]:

$$\Theta = \min_{\mathbf{c}\in\mathbf{C}} \frac{\sum\limits_{a\in\mathbf{c}} M_0(a)}{\sum\limits_{t\in\mathbf{c}} \delta(t)} \tag{2}$$

Definition 2.3. (Critical cycle and arc) A cycle $\mathbf{c}$ satisfying the equality (2) is called *critical*. An arc is called critical if it belongs to a critical cycle.

For instance in Fig. 3(c), the critical cycle is $\{d, h, j, i\}$. Efficient algorithms for computing the throughput of an EMG exist that do not require an exhaustive enumeration of all cycles [13].

2.5. Average marking

The *average marking* of an arc a, denoted as $\overline{M}(a)$, represents the average occupancy of the arc during the steady state execution. Formally the average marking vector for all arcs is defined as:

$$\overline{M} = \lim_{\tau\to\infty} \frac{1}{\tau}\int_0^{\tau} M(t)dt$$

where τ is the time variable.

Each pair $\{a, a^{\bullet}\}$ of the EMG can be seen as a simple queuing system for which Little's formula [17] can be directly applied[3]. Hence, it holds $\overline{M}(a) = \overline{R}(a)\cdot\Theta$, where $\overline{R}(a)$ is the average residence time at arc a, i.e., the average time spent by a token on the arc a [4]. The average residence time is the sum of the average waiting time due to a possible synchronization, and the average service time which in the case of EMGs is $\delta(a^{\bullet})$. Therefore the service time $\delta(a^{\bullet})$ is a lower bound for the average residence time. This leads to the inequality:

$$\overline{M}(a) \geq \delta(a^{\bullet})\cdot\Theta \qquad \text{for every arc } a \tag{3}$$

Due to the token preservation property of EMG cycles it can be proven that the token preservation property holds not only for any reachable marking, but also for the average marking. That is for any cycle $\mathbf{c}$ the sum of tokens in the initial marking and in the average marking is the same, i.e.,

$$\Sigma_{a\in\mathbf{c}} M_0(a) = \Sigma_{a\in\mathbf{c}}\overline{M}(a)$$

In particular, this statement holds for any critical cycle. Thus, equation (2) can be rewritten for a critical cycle $\mathbf{c}$ as follows:

$$\Theta = \frac{\sum\limits_{a\in\mathbf{c}} \overline{M}(a)}{\sum\limits_{t\in\mathbf{c}} \delta(t)} \tag{4}$$

[3]Little's formula holds under very general conditions. The formula $L = \lambda W$, connects the expected value L of the queue length and W, the average waiting time for a customer through λ, the arrival rate for customers eventually served.

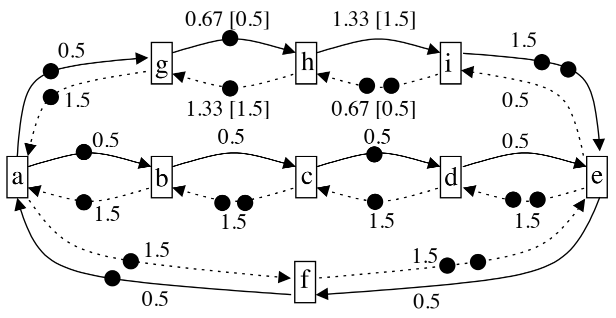

Figure 4. An EMG illustrating a tight marking.

Combining expressions (3) and (4) yields the following equation for every critical arc **a**:

$$\overline{M}(a) = \delta(a^\bullet) \cdot \Theta \tag{5}$$

Similarly, the state equation (1) can be expanded to the real domain for markings and firing vectors and is, in particular, satisfied by the average marking $\overline{M}$.

$$\overline{M} = M_0 + \mathbb{C} \cdot \sigma, \tag{6}$$
$$\text{where } \overline{M} \in \mathbb{R}^{|A|} \text{ and } \sigma \in \mathbb{R}^{|T|}$$

3. Clustering of schedulers

3.1. Tight marking

This section introduces a special marking, called *tight marking*, that facilitates the task of partitioning the initial set of transitions into clusters. Each cluster represents a set of transitions that may be collapsed into a single transition, provided that they can be activated at the same instant, without degrading the system throughput. In our scheduling setting, they can thus share the same iteration scheduler. Fork transitions (like a in Fig. 4) are potential sources of arcs with the same average markings, ab and ag, since fork transitions serve as synchronization points. A tight marking assigns, if possible, the same marking to the output arcs of fork transitions. It will be shown that a tight marking can be obtained in polynomial time by using a *linear programming* (LP) model [25].

Definition 3.1. A marking $\widetilde{M} \in \mathbb{R}^{|A|}$ is called a *tight* marking of an EMG if it satisfies:

$$\widetilde{M} = M_0 + \mathbb{C} \cdot \sigma \tag{7}$$
$$\forall\, a : \quad \widetilde{M}(a) \geq \delta(a^\bullet) \cdot \Theta \tag{8}$$
$$\forall\, t\, \exists\, a \in {}^\bullet t : \quad \widetilde{M}(a) = \delta(a^\bullet) \cdot \Theta \tag{9}$$

where $\widetilde{M} \in \mathbb{R}^{|A|}$, $\sigma \in \mathbb{R}^{|T|}$, and Θ is the throughput of the EMG. An arc a satisfying condition $\widetilde{M}(a) = \delta(a^\bullet) \cdot \Theta$ is called *tight*.

Notice that in Definition 3.1 the components of σ are not constrained to the positive real numbers. This is due to the fact that, given that $\mathbf{1}$ is a right annuller of $\mathbb{C}$, any vector $\sigma' = \sigma + j \cdot \mathbf{1}$ with $j \in \mathbb{R}$, leads to the same marking $\tilde{M}$ as σ does.

Since a tight marking satisfies (7) and (8), each critical arc a is necessarily tight. On the other hand, non-critical arcs have some slack to satisfy (7) and (8). The tight marking exploits this flexibility by adjusting the marking value for some arcs, at least one per transition (9), to the system throughput. This tight making eases the formulation of sufficient conditions to compute clusters of transitions that can share the scheduler.

Let us consider the EMG in Fig. 4. It has a single critical cycle $\{a, b, c, d, e, f\}$ with throughput equal to 0.5. Each arc in Fig. 4 is labeled with one number if its average and tight markings coincide. When they are different, the average marking is listed first and the tight marking is shown in square brackets. It can be seen that the tight marking satisfies all the conditions of Definition 3.1.

We now show how to compute a tight marking by solving an LP problem. First let us define the necessary background for linear programming.

A *linear inequality* is defined by a vector $a \in \mathbb{R}^n$ and a constant $b \in \mathbb{R}$. It is represented by $a \cdot x \leq b$ and it is *feasible* over a the reals if there exists some assignment $k \in \mathbb{R}$ to x satisfying $a \cdot k \leq b$.

A *linear programming problem* (LP) is a finite set of linear inequalities. It can be represented in matrix notation as $C \cdot x \leq B$, where each row of C corresponds to a linear inequality and B contains the constant terms of the inequalities. Optionally, it can have a linear optimization function $c^T \cdot x$ called the *objective function*. A solution of the problem is a vector that satisfies the linear inequalities. A solution is *optimal* if it maximizes the objective function (over the set of all solutions). An LP is *feasible* if it has a solution. LP can be solved in polynomial time [15]. The most popular algorithm for LP is the *simplex* [12] that performs very efficiently in practice, although it is exponential in the worst case [16].

Proposition 3.1. A tight marking of a EMG can be computed by solving the following LP problem:

$$\begin{aligned} &Maximize\ \Sigma\sigma : \\ &\qquad \delta(a^\bullet) \cdot \Theta \leq \tilde{M}(a) \quad \text{for every } a \in A \\ &\qquad \tilde{M} = M_0 + \mathbb{C} \cdot \sigma \\ &\qquad \sigma(t_c) = k \end{aligned} \tag{10}$$

where t_c is a transition that belongs to a critical cycle and k is any real constant number.

Proof: The proof is divided in two parts: a) proves that the LP is feasible and bounded; b) proves that the solution is a tight marking.

a) The set of solutions that satisfy the first two constraints is not empty, e.g., the average marking satisfies them. Since the vector $\mathbf{1}$ is a right annuller of $\mathbb{C}$, a marking $\tilde{M} = M_0 + \mathbb{C} \cdot \sigma$ can be obtained with any firing vector $\sigma' = \sigma + j \cdot \mathbf{1}$ where j is a real number. Hence, the third constraint $\sigma(t_c) = k$ can also be satisfied by the average marking. Therefore, the LP is feasible.

The third constraint forces $\sigma(t_c) = k$. Let $b \in t_c^\bullet$, then $\sigma(b^\bullet)$ cannot tend to infinity, otherwise $\tilde{M}(b)$ would tend to minus infinity and the first constraint in (10), $\delta(a^\bullet) \cdot \Theta \leq \tilde{M}(a)$ for every $a \in A$, would be violated. Since we deal with strongly connected EMGs, every arc is reachable from t_c, and therefore this reasoning can be extended to the rest of transitions of the graph.

b) Let us assume that the marking $\tilde{M}$ associated to the solution of the LP problem is not a tight marking. This implies that there exists a transition t such that for every $a \in {}^\bullet t : \delta(t) \cdot \Theta < \tilde{M}(a)$.

Notice that, in order to satisfy the first two constraints of the LP problem, each critical arc a satisfies $\delta(a^\bullet) \cdot \Theta = \tilde{M}(a)$. Therefore t is not a transition in a critical cycle, and more precisely $t \neq t_c$.

Given that $t \neq t_c$ and that for every $a \in {}^\bullet t : \delta(t) \cdot \Theta < \tilde{M}(a)$, the value of $\sigma(t)$ can be increased without violating any constraints in the LP. Such an increase produces an increase in the objective function. This is a contradiction because the objective function is maximized.

□

The first two constraints of (10) can be transformed into:

$$\delta \cdot \Theta - M_0 \leq \mathbb{C} \cdot \sigma \tag{11}$$

Since we are dealing with MGs, each row of the incidence matrix $\mathbb{C}$ contains a single positive (1) and a single negative (-1) value, while all other values are zeros. Therefore, equation (11) is a system of *difference constraints* and hence the LP (10) can be efficiently solved by the Bellman-Ford algorithm [10].

3.2. Partition of transitions into clusters

This section describes conditions that guarantee the safe merging of transitions, i.e., no throughput degrading. The set of transitions that can be merged into a single transition without performance degradation will be called *cluster*. This partition has two purposes: first, in the next section it is used to reduce considerably the initial net without degrading the performance, and second, Section 4 demonstrates that all transitions in a cluster can be assigned the same schedule.

Merging two transitions t_i and t_j in an EMG $G = (T, A, M_0, \delta)$ leads to a new EMG $G < t_i, t_j >= (T', A', M_0', \delta')$ in which two transitions t_i and t_j are replaced with a new one t_{ij}. In the new EMG, all input and output arcs of t_i and t_j of the original EMG are replaced with input and output arcs of t_{ij} such that $a \in {}^\bullet t_{ij}$ iff $(a \in {}^\bullet t_i \vee a \in {}^\bullet t_j)$ and $a \in t_{ij}^\bullet$ iff $(a \in t_i^\bullet \vee a \in t_j^\bullet)$. The initial marking for a new arc is equal to the initial marking of the corresponding replaced arc (Sect. 3.4 contains a detailed example of iterative applications of this transformation). After merging two transitions a multi-graph is obtained since two transitions can be connected by more than one arc. If the new EMG $G < t_i, t_j >$ has two identical arcs v and w, i.e., $M_0'(v) = M_0'(w)$, $L(v) = L(w)$, ${}^\bullet v = {}^\bullet w$ and $v^\bullet = w^\bullet$, then v and w can be merged into a single arc.

Definition 3.2. Transitions t_i and t_j are called *mergeable* if an EMG $G < t_i, t_j >$ obtained by merging transitions t_i and t_j in an EMG G has the same throughput as G.

The following theorem forms a basis for computing clusters in a EMG.

Theorem 3.1. Transitions t_i and t_j in an EMG are mergeable if there exist arcs $a_i \in {}^\bullet t_i$ and $a_j \in {}^\bullet t_j$ such that:

- $L(a_i) = L(a_j)$,
- $\tilde{M}(a_i) = \tilde{M}(a_j) = \delta(a_i^\bullet) \cdot \Theta$,
- $({}^\bullet a_i = {}^\bullet a_j)$ or $({}^\bullet a_i$ and ${}^\bullet a_j$ are mergeable).

Proof: Let us assume that there exist t_i, t_j that verify the conditions of the theorem. Since ${}^\bullet a_i$, ${}^\bullet a_j$ are mergeable (or ${}^\bullet a_i = {}^\bullet a_j$) we can reason on the graph obtained after merging ${}^\bullet a_i$ and ${}^\bullet a_j$ into a single transition t_x. Assume without loss of generality that $M_0(a_i) \geq M_0(a_j)$. Let us fire transition t_i as many times as $M_0(a_i) - M_0(a_j)$ producing marking M (notice that the EMGs considered in this paper are 2-bounded, and hence, it holds $M_0(a_i) - M_0(a_j) \leq 2$). Let us assume that this firing process is finite, i.e. no infinite transition firing is required (we will address this issue at the end of the proof). At M it holds $M(a_i) = M(a_j)$. Since a_i and a_j have the same input transition t_x, if t_i and t_j are merged, arcs a_i and a_j will have the same input transition and the same output transition. Thus, a_i and a_j will be identical, i.e., they will always have the same marking. In addition, the fact that $\tilde{M}(a_i) = \tilde{M}(a_j) = \delta(a_i^\bullet) \cdot \Theta$ ensures that in the steady state every arc is allowed to verify (3) after merging t_i and t_j. Therefore, after merging t_i and t_j the system throughput is preserved.

It must be taken into account that the firing of t_i may produce a marking with negative values in some of its input arcs ${}^\bullet t_i$. Since such marking is not a valid initialization, the input transitions of the arcs with negative values must also be fired. These new firings may produce a new set of arcs with negative values, and then, more firings have to be carried out.

Let us show that this firing process to avoid negative markings finishes. Notice that $M_0(a_i) - M_0(a_j) \leq 2$, therefore the markings produced by the firing process cannot assign less than -2 to any arc. On the other hand, it holds that $\tilde{M}(a_i) = \tilde{M}(a_j)$ and that $\tilde{M}(a_i') = 2 - \tilde{M}(a_i)$, where a_i' is the complementary arc of a_i. Thus, every cycle containing a_i' and a_j will have at least 2 tokens. This means the the firing process will not need to fire t_j to avoid negative markings. Moreover, since each cycle has a positive number of tokens that is preserved by any firing sequence, it will not fire any cycle of transitions. Hence, such firing process will eventually finish, and will yield a marking M' in which all arcs have non-negative values and $M'(a_i) = M'(a_j)$. □

The first two conditions of Theorem 3.1 narrow the search space to tight arcs with the same label (forward or backward). The third condition defines iterative merging. These three conditions ensure the existence of an initialization, i.e., firing sequence of transitions, that produces a marking M in which $M(a_i) = M(a_j)$ (see the proof of the Theorem 3.1). After such initialization, transitions t_i and t_j can effectively be merged. This merging will make arcs a_i and a_j identical, since $M(a_i) = M(a_j)$, $L(a_i) = L(a_j)$, ${}^\bullet a_i = {}^\bullet a_j$ and $a_i^\bullet = a_j^\bullet$, and hence they will be merged into a single arc. The set of transitions that can be merged with a given transition t, i.e. the cluster containing t, is denoted by $[t]$.

Fig. 5 shows the tight arcs of the EMG in Fig. 4. The only remaining cycle is the critical one. Transitions b and g in Fig. 4 do fulfill the conditions of Theorem 3.1 and therefore they are mergeable. Moreover, since $M_0(ab) = M_0(ag)$ transitions b and g can be merged without changing the initial state. Transitions c and h also fulfill the conditions. Given that $M_0(bc) \neq M_0(gh)$, an initialization is required before merging c and h. In this case, firing h is enough to initialize the system: such firing removes one token from gh and ih, and adds one token in hg and hi. This way, a marking M is obtained in which $M(bc) = M(gh)$ and transitions c and h can be effectively merged.

The system in Fig. 4 shows why the tight marking better captures the flexibility for merging transitions than the average marking. The arc gh is tight since $\tilde{M}(gh) = 0.5$. However it is not critical since $\overline{M}(gh) = 0.67$. Therefore, transitions c and h could not be merged if the average marking is used instead of the tight marking in Theorem 3.1.

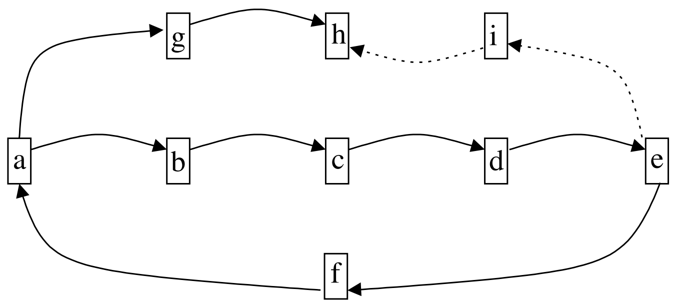

Figure 5. Tight subgraph of the system from Fig. 4.

3.3. Heuristics for reducing an EMG

In this section, we present a greedy strategy to share controllers without decreasing the system throughput. The core of the method lies in the conditions of Theorem 3.1. To check those conditions it is necessary to compute a tight marking.

When clusters have been found, the EMG can be reduced to alleviate the complexity of the model. The overall strategy for reducing an EMG involves the following steps:

1. Computation of the throughput of the system [13].
2. Computation of a tight marking (Subsection 3.1).
3. Determine sets of mergeable transitions (Theorem 3.1) by traversing the tight subgraph and merge them.
4. Determine the sets of arcs that can be merged.
5. Remove redundant arcs.

In step 3), a good heuristics is selecting critical fork transitions and exploring tight arcs at the output of these transitions. Every set of mergeable transitions then becomes a new starting point for the search of the next set and the method iterates until convergence [4]. The next subsection shows the above steps through an example.

3.4. Example

Figure 6 depicts the marked graph associated to the largest strongly connected component obtained for the logic circuit s27 of the ISCAS benchmarks [1]. For the sake of clarity only forward arcs ($L(a) = F$) are shown. The initial marking is randomly defined.

Throughput computation. The throughput of the system is 0.25. There exists only one critical cycle: $\{t_1, t_7, t_3, t_{13}\}$.

[4]Clearly, step 4) implies a change in the initial state of the system. If this must be avoided, less mergeable transitions might be found but still the approach can be applied.

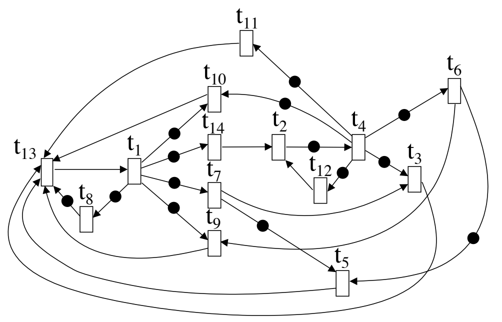

Figure 6. Circuit s27 from the MCNC benchmark suite.

Tight marking computation. Once a tight marking $\tilde{M}$ is computed, the proposed strategy visits the tight arcs, i.e., each arc a such that $\tilde{M}(a) = \Theta$. The tight arcs of the system are shown in the graph in Fig. 7. Notice that tight arcs can be both forward and backward arcs, e.g., arcs t_3t_4 and $t_{13}t_{11}$ are backward.

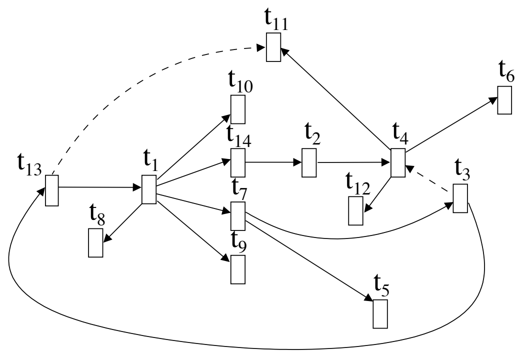

Figure 7. Tight arcs of s27.

Merging transitions. Each transition of the critical cycle is a fork in the subgraph in Fig. 7, thus, any of them can be selected to start the search for mergeable transitions. Let us pick-up t_1. Transition t_1 has five outgoing arcs t_1t_7, t_1t_8, t_1t_9, t_1t_{10} and t_1t_{14} all with the tight marking equal to the throughput and all labelled with the same label. These transitions fulfill the conditions of Theorem 3.1. Given that all those arcs have the same initial marking, transitions t_7, t_8, t_9, t_{10} and t_{14} can be merged into a single transition without decreasing the system throughput. Next, the algorithm takes the tight successor arcs of the merged transitions: $t_{14}t_2$, t_7t_3 and t_7t_5. The initial markings of those arcs are not the same: $M_0(t_{14}t_2) = M_0(t_7t_3) = 0$, while $M_0(t_7t_5) = 1$. However, transition t_5 can fire yielding a modified initial marking such that $M_0(t_{14}t_2) = M_0(t_7t_3) = M_0(t_7t_5)$. Now, transitions t_2, t_3 and t_5 can be merged.

In the next iteration, the output tight arcs of merged transitions are: t_2t_4, t_3t_{13} and t_3t_4. One of these arcs is backward (t_3t_4), hence, there is no possibility of merging its output transition t_4 with other transitions. By firing t_4 the initial markings of t_2t_4, t_3t_{13} are made equal and therefore transitions t_4 and t_{13} can be merged. Finally, the output tight arcs of t_4 and t_{13} are t_4t_6, t_4t_{11}, t_4t_{12}, $t_{13}t_1$, $t_{13}t_{11}$.

iteration	trans. merged	trans. fired
1	$t_7, t_8, t_9, t_{10}, t_{14}$	none
2	t_2, t_3, t_5	t_5
3	t_4, t_{13}	t_4
4	t_1, t_6, t_{11}, t_{12}	$2 \cdot t_6, 2 \cdot t_{11}, 2 \cdot t_{12}$

Table 1. Transitions merged in each iteration and fired to change the initial marking.

The backward arc $t_{13}t_{11}$ is discarded. In order to have the same marking on the remaining forward arcs, transitions t_6, t_{11} and t_{12} are fired twice each. Now, transitions t_1, t_6, t_{11} and t_{12} can be merged. Since all transitions have been visited no further transitions should be examined and the algorithm finishes.

Table 1 summarizes the sets of transitions merged by the algorithm during each iteration. The algorithm has merged all transitions in 4 new transitions: $a = \{t_7, t_8, t_9, t_{10}, t_{14}\}$, $b = \{t_2, t_3, t_5\}$, $c = \{t_4, t_{13}\}$ and $d = \{t_1, t_6, t_{11}, t_{12}\}$. As a result every original transition got mapped to a transition on the critical cycle. It can be checked that the result is the same if the first visited transition is other than t_1.

Merging arcs. The sets of forward arcs that can be merged after merging transitions are: $A_a = \{t_1t_{10}, t_1t_{14}, t_1t_8, t_1t_9, t_1t_7\}$, $A_b = \{t_4t_{12}, t_4t_{11}, t_4t_6, t_{13}t_1\}$, $A_c = \{t_{14}t_2, t_7t_3, t_7t_5\}$, $A_d = \{t_2t_4, t_3t_{13}\}$, $A_e = \{t_{10}t_{13}, t_9t_{13}\}$, $A_f = \{t_{12}t_2, t_6t_5\}$, $A_g = \{t_8t_{13}\}$, $A_h = \{t_4t_{10}\}$, $A_i = \{t_4t_3\}$, $A_j = \{t_{11}t_{13}\}$, $A_k = \{t_6t_9\}$, and $A_l = \{t_5t_{13}\}$. The sets of complementary arcs, $A'_a, \ldots, A'_l$, corresponding to the backward arcs and their controllers can also be merged. The net resulting of merging the mergeable transitions and merging identical arcs is shown in Fig. 8. In such figure only the forward arcs are depicted, the marking is the one obtained after initialization. Notice that in Fig. 8 there are some redundant arcs, e.g., T_dT_a arc with two tokens, that can be removed without decreasing the throughput. Nevertheless, not all redundant arcs have the same initial marking as the arcs with the same input and output transition. For this reason they explicitly need a different controller.

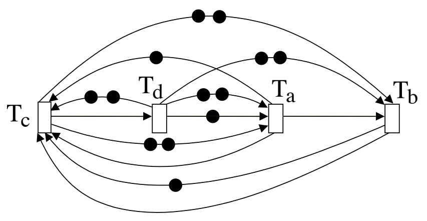

Figure 8. Net after merging transitions and arcs.

Removing redundant arcs. If two transitions are connected with multiple arcs then only the one with minimal initial marking is left. Also non-critical arcs that can be obtained as a transitive closure of other arcs are removed. Complementary arcs can be added with a marking that ensures the capacity invariant. Fig. 9 shows a final reduced EMG after merging arcs and eliminating redundant arcs. Redundant arcs can be removed using standard Petri net techniques [26].

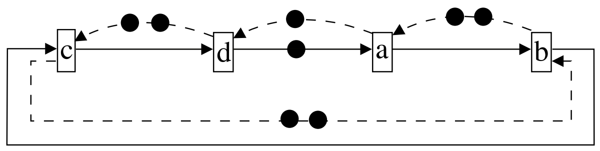

Figure 9. Reduced EMG for s27.

4. A flow for scheduling

The purpose of this section is to compute efficiently the schedule of each transition of the initial elastic system. Let us start by defining the notion of schedule:

Definition 4.1. A schedule is a binary word $w \in \{0,1\}^*$ denoted by the regular expression $w = u \cdot (v)^\omega$, with $u, v \in \{0,1\}^*$, and $(v)^\omega$ denoting infinite repetition of word v.

Given a schedule $w = u \cdot (v)^\omega$, u and v are called *transient* and *periodic* schedules, respectively. The schedule activates a given transition at instant i if the i-th bit of the schedule is a one. Once the EMG is transformed to its reduced form (as explained in Section 3.3), a simulation on the simpler EMG will assign a schedule to each transition of a cluster. An explanation on how to do this simulation can be found in [2]. It is important to stress that in our setting, the simulation is done on the reduced net, potentially requiring significantly less resources. An example of scheduling is shown in Fig. 3.

Let us now state an important property of the clusters computed in Section 3.2:

Proposition 4.1. (Properties of scheduled clusters)
Let $[t]$ be the cluster with schedule $w = u \cdot (v)^\omega$, and Θ be the throughput of the system. The following properties hold:

1. Assuming that every transition in $[t]$ is activated according to schedule w, the system has throughput Θ.
2. Every transition of the initial system S is scheduled together with some critical transition in the clustered system $S_{[t]}$.

Proof: 1. holds due to the fact that the throughput of the EMG for which the schedule is computed is Θ, and the activation of each transition in $[t]$ has been decided according to a simulation of this EMG.
2. is obtained from the following features of a tight marking: i) every arc of a critical cycle is tight, ii) the graph composed of tight arcs is connected, and iii) for every transition t, there exists $a \in {}^\bullet t$ such that a is tight.

The general strategy for scheduling presented in this paper has the following steps:

Transformation of the initial system into the EMG model: this is a crucial step that has not been considered in previous approaches. By incorporating the *dynamic* information provided by the stop signals (backward arcs in the EMG), the stall information might be explicitly transferred to the scheduling. Considering a predefined semantics (such as ASAP in [2]) is therefore unnecessary in our method. Note also that in the previous approaches, the complete equalization of the system is required, i.e. every elementary cycle of the net [9, 2] should have the same throughput as the critical cycle of the system.

Recycling/buffer sizing to achieve the throughput of the forward net: when a critical cycle contains a backward arc, it might be possible to improve the system performance by applying well-known performance optimization techniques. The goal is to attain the throughput of the initial net, i.e. to avoid throughput degradation caused by the back-pressure from stop signals. To achieve this, buffer sizing [18] or recycling [6] can be applied.

Computation of the clusters and net reduction: as explained in Section 3 and demonstrated in the current section, clusters represent an easy way of finding a good partition for both reducing the net without performance penalty and determining the sets of transitions having the same schedule. The initialization step (firing some transitions, as shown in Section 3), required for merging some transitions is crucial to guarantee that every transition can share the schedule of a transition in a critical cycle.

Simulation to find schedules for each cluster: the synchronous simulation of the reduced net will find the periodic schedules necessary for substituting the control logic in charge of the elastic protocol.

Implementation of the local controllers: The periodic schedule for each cluster can be implemented in different ways. A direct strategy is to use shift registers, as shown in Fig. 3(b). Encoding techniques for finite-state machines can be also be used to find more efficient implementations with fewer sequential elements and more logic [27]. Additionally, the handshake signals from neighboring controllers can also be used to further simplify the controllers if the periodic schedules are too long.

The scheduling strategy proposed in this paper can be seen as a method to optimize the space of global states of the system by reducing concurrency. However, this reduction is only performed in those parts in which there is slack, thus not affecting the throughput of the system. This state space reduction can be used to simplify the implementation of the controllers in various ways, either using disconnected schedulers, distributed controllers with handshakes, or hybrid approaches.

5. Layout-aware clustering

The clustering method presented in Section 3 merges transitions and arcs ignoring layout information about the circuit implementation of the corresponding elastic circuit. Hence, the strategy can result in merging elastic controllers that are separated by large physical distances in the final layout, thus creating long handshake wires that can have a negative impact on the performance of the system. It is desirable that the clustering of schedulers is performed according to the physical placement of the circuit components.

Incorporating layout information in the clustering approach can be done in two different ways.

1. The layout information can be used to restrict the merging of controllers when they are far apart, or

2. The layout unaware clustering can first be performed followed by a split of large clusters into smaller ones according to the proximity of the controllers.

By experimentation we found the latter approach more effective since it has a lower dependency on the merging order of the controllers. The method used for splitting each cluster into a set of sub-clusters is based on the well-known k-means clustering algorithm [19].

A layout-aware approach for clustering can be devised by defining the longest distance (d) between the controller and the components associated to the controller. Thus, given the longest allowed distance d, the k-means clustering algorithm produces clusters whose *smallest enclosing circle* has radius d. Checking this condition can be done in linear time [20]. The algorithm for splitting clusters is described in Algorithm 1.

Algorithm 1: Splitting of clusters

```
begin
    if radius of cluster is $<= d$ then exit
    $c \longleftarrow 2$
    repeat
        Split cluster into c sub-clusters using the $k$-means algorithm
        if radius of each subcluster is $<= d$ then exit
        $c \longleftarrow c + 1$
    until forever
end
```

In the worst case (N points that must be assigned to different clusters), the algorithm may perform N calls to the k-means algorithm, which is NP-hard. In the following section we will illustrate the significance of using placement information in the clustering of schedulers.

6. Experimental results

We have performed some experiments for the theory developed in this paper. The goal was to verify the capacity of sharing schedules with or without placement information. After this crucial step, the creation of the schedules for each cluster must be done with a technique similar to the ones presented in the literature [2, 3].

We have selected a set graphs of different sizes from the ISCAS benchmarks. The graphs were interpreted as coarse-level netlists. Each gate was interpreted as a computational block with unit delay (transition in the EMG), whereas each edge was interpreted as a communication channel (arc in the EMG). Moreover, each channel was supposed to have a half-full FIFO with capacity 2.

To generate physical information for an n-block netlist, a unit square grid with $s \times s$ cells, $s = \lceil \sqrt{n} \rceil$, was generated. All the blocks were assumed to have unit size and were placed on the layout using Capo [24]. Figure 10 depicts the placement results for the first example used in this section.

After placement, *wire pipelining* was applied to those channels longer than 10 units. In circuit design, wire pipelining is a technique to avoid communications through long wires by the insertion of intermediate buffers. In the experiments, empty FIFOs with capacity 2 were inserted in such a way that

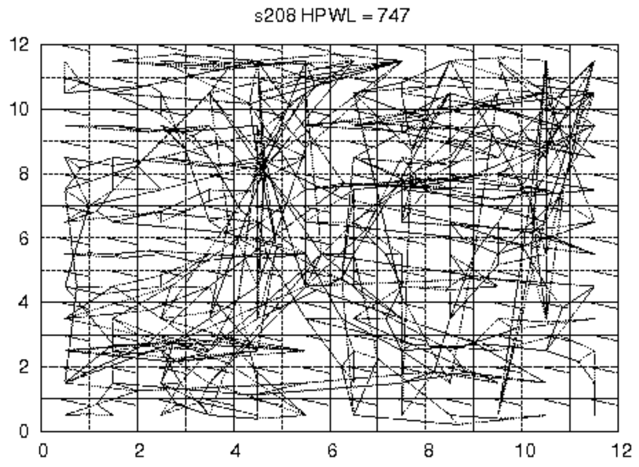

Figure 10. Placement for example s208.

the length of each channel did not exceed 10 (see Fig. 11). It is important to realize that the insertion of empty FIFOs reduces the throughput of the system when the channel is critical (i.e. it belongs to a critical cycle) due to the lack of tokens. When the channel is critical due to the absence of bubbles, the insertion of empty FIFOs only increases the latency.

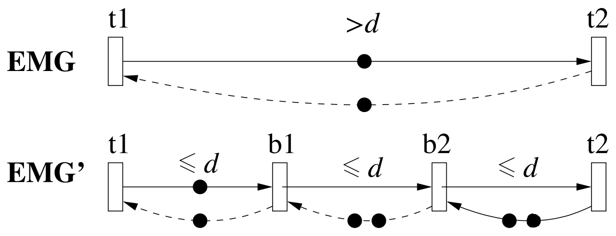

Figure 11. Wire pipelining to reduce the length of long channels.

In Table 2 columns $|T|$ and $|A|$ report the size of each benchmark, and column Θ lists its throughput after having inserted the empty FIFOs for wire pipelining. Two experiments are reported in Table 2. In columns under **Sched.** the result (number of clusters, **cls**, and arcs connecting them) of finding clusters without any restriction on the placement is shown. Clearly, this is an idealistic case that might be infeasible in many cases, but shows that in principle (as Proposition 4.1 asserts) the set of schedules for the critical cycle are enough to provide a static scheduling for the elastic system.

Columns under **Place & Sched.** show a more realistic experiment, when the information of placement is taken into account to avoid clusters containing nodes for which the schedule register might be too far. The k-means algorithm is used for cluster splitting when the distance inside the cluster is greater than 10 units.

				Sched.		Place & Sched.		
example	$\|T\|$	$\|A\|$	Θ	**cls**	**arcs**	**cls**	**arcs**	**CPU**
s208	155	450	0.50	2	6	2	6	$<$ 1 s.
s344	270	764	0.50	2	6	4	24	1 s.
s420	321	954	0.50	4	8	8	48	$<$ 1 s.
s444	338	1060	0.50	4	8	8	54	$<$ 1 s.
s510	446	1990	0.50	4	8	6	20	1 s.
s713	505	1360	0.33	3	15	12	142	1 s.
s820	325	2220	0.50	4	8	7	39	$<$ 1 s.
s832	1121	4686	0.33	3	15	9	102	1 s.
s838	743	2198	0.33	5	17	14	170	1 s.
s953	925	2688	0.25	4	28	18	365	1 s.
s1423	1079	3252	0.25	4	28	26	475	1 s.
s1488	1127	5718	0.50	4	8	10	62	10 s.
s1494	1239	6220	0.50	4	8	12	86	17 s.
s5378	4324	12160	0.16	6	66	124	2493	1m.
s9234	4219	11200	0.14	99	249	233	2531	1 m.

Table 2. Experimental results.

7. Conclusions

This paper has shown that elasticity can be incorporated in conventional circuits at the expense of a small control cost. While still having distributed controllers along the system, the network can be significantly reduced by clustering controllers.

The methods presented reduce the complexity of the control layer necessary in an elastic design. Iteration schedulers are used to mimic the stalling signals controlling the data flow. By replacing the controllers and wires implementing the handshake protocols by iteration schedules, no control communication is needed and routing and placement constraints are alleviated. In a higher optimization level, an efficient method for clustering is presented for further optimization, allowing several computational blocks to share the same schedule.

Acknowledgements

This work has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

References

[1] ISCAS benchmark home page. http://www.cerc.utexas.edu/itc99-benchmarks/bench.html.

[2] J. Boucaron, J. Millo, and R. de Simone. Formal methods of scheduling for latency-insensitive designs. *EURASIP journal on Embedded Systems*, 2006.

[3] F. R. Boyer, E. M. Aboulhamid, Y. Savaria, and M. Boyer. Optimal design of synchronous circuits using software pipelining techniques. *ACM Trans. Design Autom. Electr. Syst.*, 6(4):516–532, 2001.

[4] J. Campos and M. Silva. Structural Techniques and Performance Bounds of Stochastic Petri Net Models. In *Advances in Petri Nets 1992*, volume 609 of *LNCS*. Springer, 1992.

[5] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design*, 20(9):1059–1076, Sept. 2001.

[6] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Performance analysis and optimization of latency insensitive systems. In *Proc. ACM/IEEE Design Automation Conference*, pages 361–367, June 2000.

[7] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin. Elastic circuits. *IEEE Transactions on Computer-Aided Design*, 28(10):1437–1455, Oct. 2009.

[8] J. Carmona, J. Júlvez, J. Cortadella, and M. Kishinevsky. Scheduling synchronous elastic design. In *9th International Conference on Application of Concurrency to System Design (ACSD 2009)*, pages 52–59, 2009.

[9] M. Casu and L. Macchiarulo. A new approach to latency insensitive design. In *Proc. Digital Automation Conference (DAC)*, pages 576–581, June 2004.

[10] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[11] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proc. ACM/IEEE Design Automation Conference*, pages 657–662, July 2006.

[12] G. Dantzig. *Linear programming and extensions*. Princeton University Press, Princeton NJ, 1963.

[13] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Transactions on Computer-Aided Design*, 17(10):889–899, 1998.

[14] A. Edman and C. Svensson. Timing closure through a globally synchronous, timing partitioned design methodology. In *DAC*, pages 71–74, 2004.

[15] L. Khachian. A polynomial algorithm in linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979.

[16] V. Klee and G. Minty. How good is the simplex algorithm. In *Inequalities III*, pages 159–172. Academic Press, New York, 1972.

[17] J. D. C. Little. A proof of the queueing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[18] R. Lu and C.-K. Koh. Performance optimization of latency insensitive systems through buffer queue sizing of communication channels. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 227–231, Nov. 2003.

[19] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symp. on Mathematical Statistics and Probability*, volume 1, USA, 1967.

[20] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12(4), 1983.

[21] J.-V. Millo. *Ordonnancements priodiques dans les rseaux de processus : Application la conception insensible aux latences*. PhD thesis, Universit de Nice-Sophia Antipolis, December 2008.

[22] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, Apr. 1989.

[23] C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Trans. Software Eng.*, 6(5):440–449, 1980.

[24] J. A. Roy, D. A. Papa, S. N. Adya, H. H. Chan, A. N. Ng, J. F. Lu, and I. L. Markov. Capo: robust and scalable open-source min-cut floorplacer. In *ISPD '05*, USA, 2005.

[25] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[26] M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In Reisig, W. and Rozenberg, G., editors, *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, volume 1491, pages 309–373. Springer-Verlag, 1998.

[27] T. Villa, T. Kam, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. *Synthesis of Finite State Machines: Logic Optimization*. Kluwer Academic Publishers, 1997.