Scheduling Synchronous Elastic Designs

Josep Carmona
Universitat Politècnica de Catalunya
Barcelona, Spain

Jorge Júlvez
Universidad de Zaragoza
Zaragoza, Spain

Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

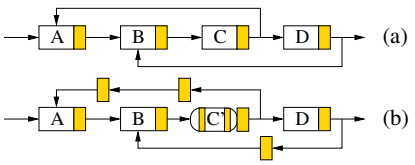
Michael Kishinevsky
Intel Corporation
Hillsboro, USA

Abstract

Asynchronous and latency-insensitive circuits offer a similar form of elasticity that tolerates variations in the delays of communication resources of a system. This flexibility comes at the expense of including 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., [4, 8, 10]). 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 part (a) of the previous figure, 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 1-cycle delay. If for scalability reasons, when migrating to a new technology, the long wires $C \rightarrow A$ and $D \rightarrow B$ no longer can produce the communication within one clock cycle, intermidiate registers must 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 figure (b) above, 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, this problems are avoided if the elastic paradigm 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 use in coordination with the clock. This elasticity comes at the price of extra area and a reduction of the routability zones, given that handshake wires wrap all the circuit area.

Scheduling latency-insensitive designs [1, 2, 6] is a circuit-level transformation that replaces the handshake controllers in each combinational block by iterative schedulers (e.g. shift registers with the activation pattern for each block), thus eliminating the need for the handshake signals. 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, scheduling might be an elegant alternative.

Some attempts for applying scheduling to latency-insensitive circuits have been done in the literature. In [6], a scheduling technique is presented for simplifying latency insensitive designs, based on a previous technique for scheduling using software pipelining techniques [2]. Another approach is presented in [1]. 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 [6], the clock is divided into multiphases

to ensure the equalization (time), while [1] 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 [1]) that requires the activation of a computation node as soon as the inputs are available.

1.1 Example and contributions

The main contributions of this paper are the following:

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 constraining (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.

In circuits where the maximal throughput cannot be achieved due to stalls, performance optimization techniques are applied before computing schedules. This is illustrated in the example of Fig. 1. A fragment of the real elastic system is shown in Fig. 1(a). The initial model, shown as a Petri net in Fig. 1(c), models the behavior of the system where transitions (boxes) represent combinational logic in stages, arcs denote latches and bullets in arcs represent the initial data distribution. The model is augmented with stalling information where *backward arcs*, denoted in discontinuous lines, are used to make the stalling information explicit. The model in Fig. 1(c) 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. 1(d) shows a possible transformation (called *recycling* [5]) 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. 1(d). These schedules replace controllers and the wires corresponding to handshake signals, as shown in Fig. 1(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 reduced. In the example of Fig. 1, our technique will find that the registers for b, f and i can share the same iteration scheduler, and hence only one instead

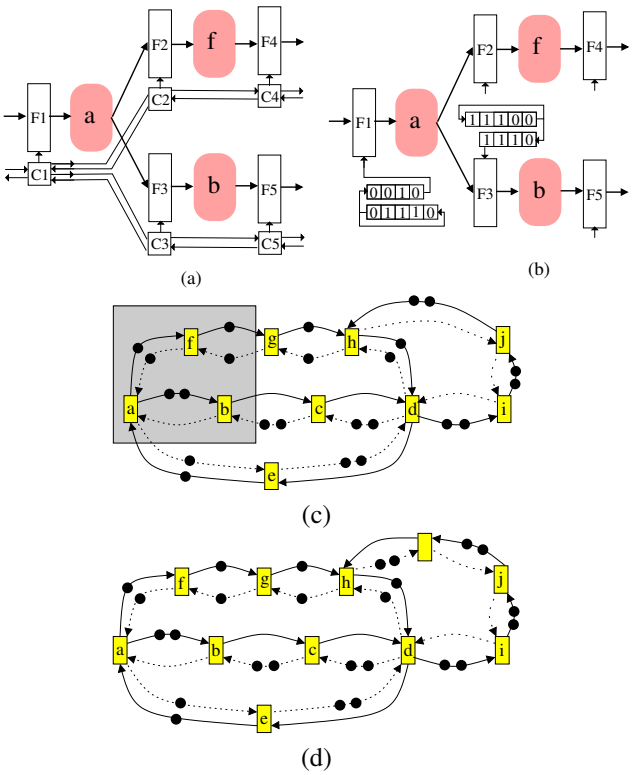


Figure 1. (a) Fragment of an elastic system (Ci are controllers, Fi are latches, a, b and f denote combinational logic), (b) schedule replacing the control layer (only shown the registers for a and b), (c) Initial Petri net model extended with stall information for the complete system (grey box represents the part depicted in (a)-(b)), (d) transformation to acquire optimal performance.

of three are 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 placement/routing constraints. We have implemented a placement-aware partition technique that takes into account this information.

The remainder of this paper is organized as follows: in Section 2 we provide 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 merged 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 [13] for a survey on Petri Nets.

2.1 Timed Marked Graphs

Definition 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 is a set of directed arcs, $M_0 : A \rightarrow \mathbb{N}$ is a marking that assigns an initial number of tokens to each arc, and $\delta : T \rightarrow \mathbb{R}$ 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 $t_i, t_j \in T$.

2.2 State equation and token preservation

Some useful basics of strongly connected MGs are [13]:
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).
Token preservation. If the marking M is reachable then $\Sigma_{a \in \mathbf{c}} M(a) = \Sigma_{a \in \mathbf{c}} M_0(a)$ for every cycle $\mathbf{c}$ of the MG.

2.3 Elastic Marked Graphs

Definition 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. 1(c)). 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 an 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 system. For instance, the FIFO represented by the arc pair $\{c, d\}$ in Figure 1(c) is 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.

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 [14]:

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

Definition 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. 1(c), the critical cycle is $\{d, h, j, i\}$. Many efficient algorithms for computing the throughput of an EMG exist that do not require an exhaustive enumeration of all cycles [9].

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 [11] can be directly applied[1]. 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 [3]. 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:

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

Combining expressions (3) and (4) yields the following equation for every critical arc $\mathbf{a}$:

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

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

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

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. 2) are potential sources of arcs with 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 means of a linear programming problem.

[1]Little's formula can be applied to any general queuing system. 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.

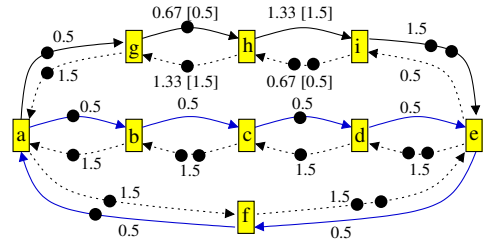


Figure 2. An EMG illustrating a tight marking.

Definition 4 *A marking $\tilde{M} \in \mathbb{R}^{|A|}$ is called a* tight *marking of an EMG if it satisfies:*

$$\tilde{M} = M_0 + \mathbb{C} \cdot \sigma \tag{7}$$

$$\forall\, a: \quad \tilde{M}(a) \geq \delta(a^\bullet) \cdot \Theta \tag{8}$$

$$\forall\, t\, \exists\, a \in {}^\bullet t: \quad \tilde{M}(a) = \delta(a^\bullet) \cdot \Theta \tag{9}$$

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

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. 2. It has a single critical cycle $\{a, b, c, d, e, f\}$ with a throughput 0.5. Each arc in Fig. 2 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 4.

Proposition 1 *A tight marking of a EMG can be computed by solving the following Linear Programming (LP) problem:*

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

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

Proof: See Appendix.

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 [7].

3.2 Partition of transitions into clusters

This section describes conditions that guarantee the safe, i.e., no throughput degrading, merging of transitions. 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}. All input and output arcs of t_i and t_j 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. 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 5 *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 2 *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: See Appendix.

The first two conditions of Theorem 2 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 2 in the Appendix). After such initialization, transitions t_i and t_j can effectively be merged. This merging will make arcs a_i and a_j be 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. 3 shows the tight arcs of the EMG in Fig. 2. The only remaining cycle is the critical one. Transitions b and g in Fig. 2 do fulfill the conditions of Theorem 2 and therefore they are mergeable. Moreover, since $M_0(ab) = M_0(ag)$ transitions b and g can be merged without retiming 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. 2 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 get merged if the average marking is used in place of the tight marking in Theorem 2.

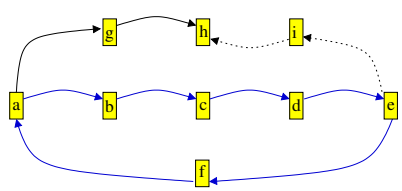


Figure 3. Tight subgraph of the system from Fig. 2.

3.3 Heuristics for reducing an EMG

When clusters have been found, the EMG can be reduced to alleviate the complexity of the model. The overall strategy involves the following steps: 1) Computation of the system throughput [9], 2) computation of a tight marking (Subsection 3.1), 3) determine the sets of mergeable transitions

(Theorem 2) by traversing the tight subgraph, 4) fire transitions to obtain the same marking in the input arcs of the mergeable transitions, and 5) merge mergeable transitions and identical 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 the fixed point [2].

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 6 *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 [1]. 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. 1.

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

Proposition 3 (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. *The system considering that every transition in $[t]$ is activated according to schedule w 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.

[2] 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.

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

Transform 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. This makes unnecessary to consider predefined semantics (e.g. *ASAP* in [1]). It is also remarkable the fact that in previous approaches, the complete *equalization* of the system is required, meaning that the throughput of the system is required *for every elementary cycle of the net* [1, 6]. This restriction is no longer required in the approach presented in this paper.

Recycling/buffer sizing to achieve the throughput of the forward net: when some 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 [12] or recycling [5] can be applied.

Computation of the clusters and net reduction: as explained in Section 3 and demonstrated in this 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.

5 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

				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.
s713	505	1360	0.33	3	15	12	142	1 s.
s832	1121	4686	0.33	3	15	9	102	1 s.
s953	925	2688	0.25	4	28	18	365	1 s.
s1423	1079	3252	0.25	4	28	26	475	1 s.
s5378	4324	12160	0.16	6	66	124	2493	1m.

Table 1. Experimental results.

the ones presented in the literature [1, 2].

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 [15]. After placement, wire pipelining was applied to those channels longer than 10 units. Empty FIFOs with capacity 2 were inserted in such a way that the length of each channel did not exceed 10. 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.

In Table 1 columns $|T|/|A|$ report the size of the examples, and column Θ reports the throughput of each example after having inserted the empty FIFOs for wire pipelining. Two experiments are reported in Table 1. 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 3 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 classes 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.

6 Conclusions and Open problems

The methods presented in this paper 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.

A future line of investigation is how to integrate the theory presented in this paper together with the use of variable latency units. It is clear that schedules assume a static functioning of the system, and therefore the activation bits produced by the schedulers can not be delayed when the unit takes longer than expected, because this local blocking may affect to adjacent schedulers and so on, thus creating a global effect that may corrupt the functioning of the system. A possible solution would be to use maximal delays when computing the schedules, but then the system will be working in a sub-optimal manner. Another possibility is to use different schedules for different latencies per unit (activated by means of a multiplexer and extra logic), but then it is not clear whereas it is worth the substition of the initial control implementing the handshake signals by this complicated mechanism.

Acknowledgements

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

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

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

[12] 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.

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

[14] 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.

[15] 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.

Appendix

Proof of Proposition 1:
Since EMG s are repetitive systems, a marking M can be reached with any firing sequence $\sigma + j \cdot \mathbf{1}$ where j is any real number. A constraint like $\sigma(t_a) = k$ ensures boundedness of the solution. Since t_a is in a critical cycle, there exists an arc $a \in {}^\bullet t_a$ such that a is also in the same critical cycle. Hence, the solution of the LP will necessarily verify $\tilde{M}(a) = \delta(a^\bullet) \cdot \Theta$ (see equation (5)). Given that the objective function $\Sigma\sigma$ is maximized, for every transition t there will exist $a \in {}^\bullet t$ such that $\tilde{M}(a) = \delta(t) \cdot \Theta$. Hence, the obtained marking $\tilde{M}$ is a tight marking. □

Proof of Theorem 2:
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 joinable (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 consider in this paper are 2-bounded, and hence, it holds $M_0(a_i) - M_0(a_j) \leq 2$). 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. Given that $\tilde{M}(a_i) = \tilde{M}(a_j)$ and $\tilde{M}(a_i') = 2 - \tilde{M}(a_i)$, where a_i' is the complementary arc of a_i, any of the cycles containing a_i' and a_j will have at least 2 tokens. This implies that the firing process to avoid negative markings (the lowest possible value is -2 since $M_0(a_i) - M_0(a_j) \leq 2$) will not fire transition t_j. 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 $M'(a_i) = M'(a_j)$. □