Elasticity and Petri nets

Jordi Cortadella[1], Michael Kishinevsky[2],
Dmitry Bufistov[1], Josep Carmona[1], and Jorge Júlvez[1]

[1] Universitat Politècnica de Catalunya, Jordi Girona, 1-3, 08034, Barcelona, Spain
[2] Intel Corporation, 2111 NE 25th Ave., Hillsboro, OR 97124, USA
jordicf@lsi.upc.edu,michael.kishinevsky@intel.com,
{dmitry,jcarmona,julvez}@lsi.upc.edu

Abstract. Digital electronic systems typically use synchronous clocks and primarily assume fixed duration of their operations to simplify the design process. Time elastic systems can be constructed either by replacing the clock with communication handshakes (asynchronous version) or by augmenting the clock with a synchronous version of a handshake (synchronous version). Time elastic systems can tolerate static and dynamic changes in delays (asynchronous case) or latencies (synchronous case) of operations that can be used for modularity, ease of reuse and better power-delay trade-off. This paper describes methods for the modeling, performance analysis and optimization of elastic systems using Marked Graphs and their extensions capable of describing behavior with early evaluation. The paper uses synchronous elastic systems (aka latency-tolerant systems) for illustrating the use of Petri Nets, however most of the methods can be applied without changes (except changing the delay model associated with events of the system) to asynchronous elastic systems.

1 Introduction

Synchronous systems dominate digital design practices in the areas of electronic system design and embedded systems. Such systems assume the presence of a global time reference - global clock - which significantly simplifies design tasks and enable usage of zero delay abstraction for computation and communication delays. When designing or analyzing a digital synchronous circuit, one implicitly assumes the existence of a master clock that determines the frequency at which computations are performed and input/output data are transferred.

The specification of synchronous systems typically rely on precise knowledge on latencies (i.e., delays as measured in number of clock cycles) of different computations. Such knowledge, that is typically required from early stages in design specifications, may make the design process highly inflexible to possible changes in communication and computation latencies or delays. In addition, it restricts the usage of adaptive, variable delay or latencies of components since static scheduling of such components is a much harder (or impossible) job and typically complicates the system description.

In contrast, these assumptions do not apply to software programs or distributed communication over Internet, for which one assumes that the response time will depend on a variety of factors beyond the control of the user: the current workload of the operating system, the cache hit ratio, the traffic on the network, etc.

This is a post-peer-review, pre-copyedit version of an article published in *LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*. The final authenticated version is available online at: http://dx.doi.org/10.1007/978-3-540-89287-8_13

One could say that software programs and Internet communication are elastic, since they can adapt themselves to the specific characteristics of the resources required to execute them and to the environment that interacts with them.

With current and future nanotechnologies, circuits resemble more a distributed network of devices with variable computation and communication delays. For example, a factor like the temperature of a specific region of a chip may change the frequency of a local clock and the response time of a particular functional unit. However, conventional circuits are often not designed in a way that allows changing the timing behavior of some components arbitrarily without modifying the functional behavior of the system.

For several decades researchers have studied systems that are tolerant to the variability of different parameters of a circuit: delay, power supply, temperature. One line of research (that was used in many industrial designs) adopts frequency of the clock and voltage levels to changing operational parameters. The other natural way of improving the tolerance to variability in delays is to eliminate the clock from the system, making the entire system asynchronous.

1.1 Two Forms of Elastic Systems

Like a distributed network, the components of an asynchronous circuit talk to each other by means of handshake signals that commit to some protocol. Typically, there is a local bi-directional synchronization for each pair of components that must exchange data. In its minimal form, the synchronization is implemented by a pair of signals called request and acknowledge.

The term "elastic circuit" initially referred to pipelines that were tolerant to the variability of input data arrival and computation delays. For example, Ivan Sutherland [1] used the term elasticity in his Turing award lecture on micropipelines.

Asynchronous systems [2–5] imply additional design complexity, since they often encode information in signal transitions. Therefore, the asynchronous circuit must not produce glitches or other transient signal transitions that could result in misinterpretations of the information. It is important to entirely avoid spurious transitions (also called glitches or hazards) or to restrict glitches to the timing intervals during which the signal is not observed. Both constraints make the design of asynchronous circuits considerably more challenging than the synchronous one.

For this reason, several research efforts limit the elasticity of asynchronous systems to discrete multiples of a certain time interval, e.g., the period of a synchronous clock. Since the mid-90's, this idea has evolved and reappeared in different forms under several names, such as synchronous emulation of asynchronous circuits [6], synchronous handshake circuits [7], latency-insensitive design [8,9] or synchronous elastic systems [10–12]. In all these variants, the systems can tolerate changes to latencies of components, but events are synchronized to a common clock.

A synchronous elastic system resembles a conventional clock circuit, but every data item in it has an associated valid bit. Every functional unit can also issue a stop bit to stall the activity of the neighboring units when it is not ready to receive information. These bits implements a synchronous version of the handshake protocol optimized in comparison with an asynchronous request/acknowledge protocol thanks to the presence of the clock reference.

By incorporating synchronicity, the design of elastic systems becomes easier. As in regular synchronous circuits, signals must stabilize only by the end of the clock period and are allowed to have glitches. Therefore, the existing infrastructure and methods for synchronous design can be re-used for synchronous elastic circuits.

Elastic circuits pose new opportunities and challenges in the design of future digital systems. Their tolerance to variable latency motivates the design of functional units optimized for the most frequent cases (instead of the worst case), offering a better average delay and new design trade-offs. They enable dynamic changes in latencies (in a synchronous case) or delays (in the asynchronous case) and dynamic adaptation to different environmental scenarios (temperature, power supply, clock frequency, etc). Layout synthesis can benefit from elasticity, since elasticity can be introduced into layout with few incremental changes enabling fine-tuning of the system for better power and performance. Elasticity introduces a certain degree of dynamic scheduling into system behavior making the optimal scheduling a more challenging problem. It also allows for new dimensions in high-level optimization and transformations.

1.2 Use of Petri Nets for Modeling Elastic Systems

It was discovered during the MIT MAC project [13–15] that Petri Nets, with their capabilities for describing distributed asynchronous computations as collection of asynchronous concurrent behaviors, is a natural way of specifying asynchronous pipelined systems. Such description can then be used for performance analysis, synthesis, validation and other forms of formal reasoning. This line of research was further explored later by multiple research groups.

In this paper we will illustrate how Petri Nets can be used for modeling Synchronous Elastic Systems (ES). The reader should keep in mind that we have chosen the synchronous version primarily for illustrative purpose and that methods for modeling of elastic systems with marked graphs (Section 3.1), for slack matching and buffer sizing (Section 4), and for performance modeling of systems with early evaluation (Section 6) can be applied equally well to the asynchronous implementation. The only adjustment that would be then required is to change the delay annotation of the Petri Nets events with other forms of delays (e.g., with real delay numbers to model continuous time domain instead of integers used for discrete time domain in synchronous systems). Methods for control optimization (Section 5) and for retiming and recycling (Section 7) rely on the synchronous nature of the systems.

Petri nets have been extensively used in asynchronous circuit design. In [16, 17], marked graphs are the underlying formalism to model the flow of data in asynchronous circuits. Signal Transition Graphs [18, 19] have also been used to specify asynchronous controllers. Several examples and areas illustrating synergies between hardware and Petri nets can be found in [20].

In this paper we focus exclusively on the use of Marked Graphs and their new extension for modeling systems with early evaluation. This is because synchronous elastic systems can be adequately modeled with this sub-classes of Petri Nets that is much easier to analyze and use for formal reasoning. When analyzing the performance, we will often assume that these systems are composed of equally-timed units

(e.g. 1-cycle delays). This assumption is not a limitation, but just a simplification to improve the readability of the paper. The reader will soon realize that many of the methods discussed in the paper can be easily extended to units with different delays.

Most of the strategies use either linear or mixed-integer linear programming (MILP) to solve the stated problems. Linear programming (LP) problems can be solved in polynomial time [21], while mixed-integer linear programming problems are NP-complete problems for which several reliable solvers exist [22].

2 Elastic Systems

2.1 Introduction

Synchronous circuits are often modeled, at a certain level of abstraction, as machines that read inputs and write outputs at every cycle. The outputs at cycle i are produced according to a calculation that depends on the inputs at cycles $0, \ldots, i$. Computations and data transfers are assumed to take zero delay.

Latency-insensitive design [8] aims at relaxing this model by elasticizing the time dimension and decoupling the cycles from the calculations of the circuit. It enables the design of circuits tolerant to any discrete variation (in the number of cycles) of the computation and communication delays. With this modular approach, the functionality of the system only depends on the functionality of its components and not on their timing characteristics. The motivation for latency-insensitive design comes from the difficulties with timing and communication in nanoscale technologies. The number of cycles required to transmit data from a sender to a receiver is determined by the distance between them, and often cannot be accurately known until the chip layout is generated late in the design process. Traditional design approaches require fixing the communication latencies up front, and these are difficult to amend when layout information finally becomes available. Elastic circuits offer a solution to this problem. In addition, their modularity promises novel methods for microarchitectural design that can use variable-latency components and tolerate static and dynamic changes in communication latencies, while - unlike asynchronous circuits - still employing standard synchronous design tools and methods.

Figure 1(a) depicts the timing behavior of a conventional synchronous adder that reads input and produces output data at every cycle (boxes represent cycles). In this adder, the i-th output value is produced at the i-th cycle. Figure 1(b) depicts a related behavior of an elastic adder- a synchronous circuit too - in which data transfer occurs in some cycles and not in others. We refer to the transferred data items simply as data and we say that idle cycles contain bubbles.

Elasticization decouples cycle count from data count. In a conventional synchronous circuit, the i-th data of a wire is transmitted at the i-th cycle, whereas in a synchronous elastic circuit the i-th data is transmitted at some cycle $k \geq i$.

Turning a conventional synchronous adder into a synchronous elastic adder requires a communication discipline that differentiates idle from non-idle cycles (bubbles from data). This communication is usually supported by a pair of wires that synchronizes the sender and the receiver.

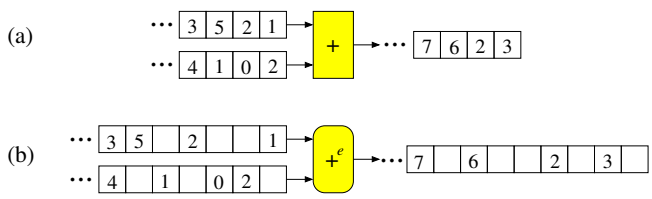


Fig. 1. (a) Conventional synchronous adder, (b) Synchronous elastic adder.

In asynchronous circuits, synchronization is typically implemented by two wires called *request* (from sender to receiver) and *acknowledge* (from receiver to sender). In synchronous circuits, different nomenclatures have been used. In this paper we will call *valid* the wire from sender to receiver that indicates the validity of the data. We will also call *stop* the wire from receiver to sender that, when asserted, indicates that the receiver has not been able to accept data.

Different synchronization protocols for elasticity can be defined. In this paper we will focus on a specific one called SELF [11] (Synchronous Elastic Flow). This protocol has been inspired on the theory of latency-insensitive design [8] and in some implementations of synchronous elastic pipelines [10].

In SELF, every input or output wire X in a synchronous component is associated to a *channel* in the elastic version of the same component. The channel is a triple of wires $< X, \mathsf{valid}_X, \mathsf{stop}_X >$, with X carrying the data and the other two wires implementing the control bits, as shown in Figure 2(b). Data is transferred on this channel when $\mathsf{valid}_X = 1$ and $\mathsf{stop}_X = 0$: the sender sends valid data and the receiver is ready to accept it.

Since elastic networks tolerate any variability in the latency of the components, empty FIFO buffers can be inserted in any channel, as shown in Figure 2(b), without changing the functional behavior of the network.

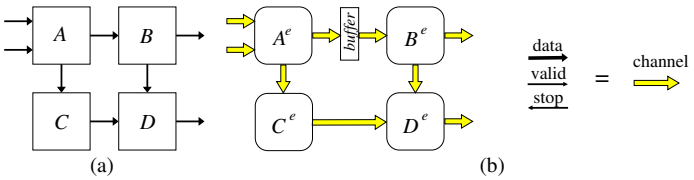


Fig. 2. (a) Synchronous network, (b) its elastic counterpart.

2.2 Architectural View of Elastic Circuits

The FIFO buffers referred in the previous section will be called *Elastic Buffers* (EB). In elastic systems, the capacity of EBs has a direct impact on the performance. For an implementation of elasticity based on distributed control between neighboring blocks,

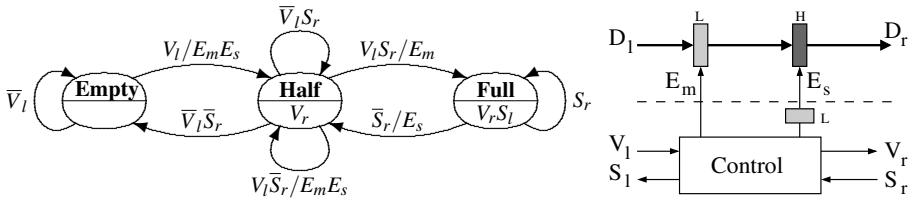


Fig. 3. Specification of the latch-based EB.

EBs must have a capacity greater than one slot to avoid a degradation in performance. In particular for one-cycle propagation latency in the forward and backward directions, it has been proved that EBs with a capacity of two slots can guarantee the same performance as a non-elastic system [8].

There are different ways of implementing EBs. In [11], a latch-based implementation of EBs was proposed, in which each FIFO with capacity two was implemented with a pair of *Elastic Half-Buffers* (EHB). An EHB consists of a transparent latch and an associated handshake controller. An EB is composed of two EHBs in a similar way as flip-flops are implemented as a pair of transparent latches with opposite polarity (master and slave).

Figure 3 depicts the FSM specifications this scheme, where V and S represent the *valid* and *stop* signals of the handshakes and E represents the *enable* signal of the latch (transparent when high). The latches are labelled with the phase of the clock, L (active low) or H (active high). To simplify the drawing the clock lines are not shown. The enable signals must be AND-ed with the corresponding clock phase for a proper operation.

An enable signal for transparent latches must be emitted on the opposite phase and be stable during the active phase of the latch. Thus, the E_s signal for the slave latch is emitted on the L phase.

The FSM specification of Figure 3 is similar to the specification of a 2-slot FIFO: in the *Empty* state no valid data is captured in the data-path, in the *Half-full* state, an output slave latch keeps valid data, in the *Full* state - both latches keep valid data and the EB requests the sender to stop.

Let us show an architectural example of an elastic communication, with the circuit of Figure 4. It represents part of a circuit where a sender provides data to a receiver. It is assumed a long distance between them, so elastic buffers are inserted accordingly as shown in the figure. To make the example more general, data is processed between the latches (boxes named *C*L). White boxes represent the control part for each one of the EHBs. The example contains consecutive snapshots (left to right, top to bottom) of the consecutive states of the elastic circuit when the communication is taking place.

The situation initially is the following (top-left configuration in the figure): all but the second latch hold valid data, shown by the circles inside them. The valid bits are 1 in all the stages. The receiver is blocked, hence it has set the stop bit to 1, which has propagated two stages further towards the sender. The sender is not aware yet of the blocking of the receiver, due to the incoming stop bit with value 0. The next phase of the clock is H, so the next configuration contains the transmission into the second

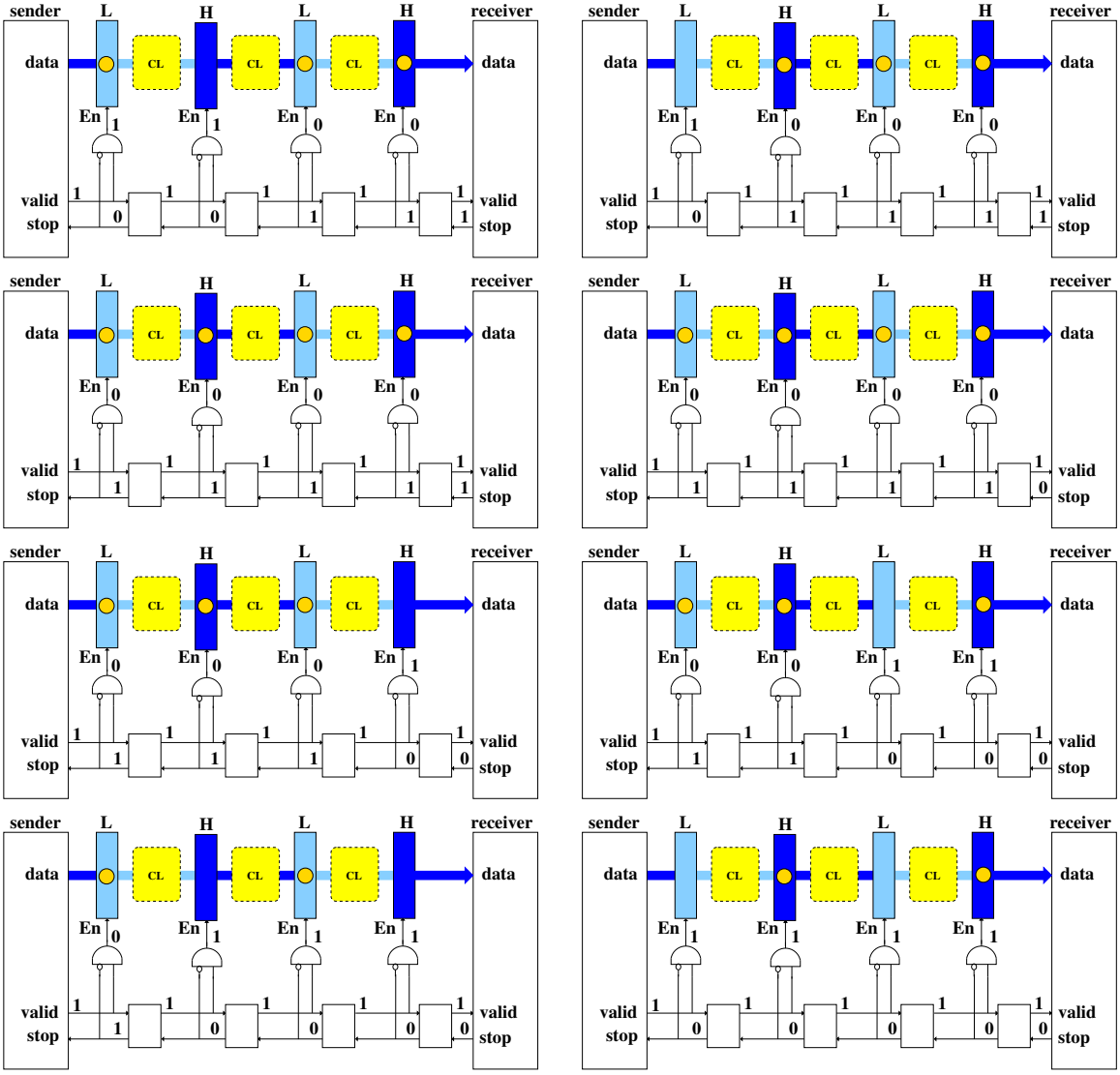


Fig. 4. Simulation of an Elastic circuit.

latch (labeled with *H*) of the data from the first *L* latch. Additionally the stop bit has travelled one stage further towards the sender. The next phase of the clock (low) allows the sender to store data in the first *L* latch, and the stop bit has reached the sender. Stop bits are also known as back-pressure. After this phase, the whole channel is blocked with data not processed by the receiver. New data coming from the sender must wait an arbitrary amount of time until the receiver is able to process the data on this channel. The forthcoming configurations in the figure (from the fourth to the eighth) show how

the channel becomes available again when the receiver starts processing data from the full channel. Along consecutive stages, the unset stop bit travels towards the sender and the latches become enabled again.

3 Marked Graph Models for Synchronous Elasticity

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 [23] for a survey on Petri Nets.

3.1 Marked Graphs

Definition 1. *A* Marked Graph *(MG) is a tuple $G = (T, A, M_0)$, where T is a set of transitions (also called nodes), $A \subseteq T \times T$ is a set of directed arcs, and $M_0 : A \to \mathbb{N}$ is a marking that assigns an initial number of tokens to each arc.*

Without loss of generality, we model elastic systems with strongly connected MGs. 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.

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

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

A transition t is *enabled* at a marking M if $M(a) > 0$ for every $a \in {}^\bullet t$. Any enabled transition t can fire. The firing of t removes one token from each input arc of t, and adds one token to each output arc of t.

Definition 2 (Reachability). *A marking M is said to be* reachable *from M_0 if there is a sequence of transitions that can fire starting from M_0 and leading to M.*

Definition 3 (Liveness). *An MG is said to be live if every node can eventually fire from any reachable marking.*

For the sake of notation, the total number of tokens in a subset $\phi \subseteq A$ at a given marking M is denoted by $M(\phi) = \sum_{a \in \phi} M(a)$. Some useful properties of strongly connected MGs [23] are:

Property 1 (Liveness). *An MG is* live *iff every cycle* $\mathbf{c}$ *is marked positively at M_0, i.e., $M_0(\mathbf{c}) > 0$.*

All the MGs considered throughout this paper are assumed to be live.

Property 2 (State equation and reachability). *A marking $M \geq 0$ is reachable from the initial marking M_0 iff the state equation*

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

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).

Property 3 (Cycles and reachability). *A marking M is reachable iff $M(\mathbf{c}) = M_0(\mathbf{c})$ for every cycle $\mathbf{c}$ of the MG.*

3.2 Timed Marked Graphs

Definition 4. *A* Timed Marked Graph *(TMG) is a tuple $G = (T, A, M_0, \delta)$, where (T, A, M_0) is a MG, and $\delta : T \rightarrow \mathbb{R}^+ \cup \{0\}$ assigns a non-negative delay to every transition.*

In a TMG, a transition t fires $\delta(t)$ time units after becoming enabled. In order to correctly model the time behavior of the circuits, single server semantics is adopted, i.e., no multiple instances of the same transition can fire simultaneously. Notice that single server semantics is a particular case of infinite server semantics: the addition of a self-loop place with one token, i.e., a place p such that $p^\bullet = {}^\bullet p$ and $M_0(p) = 1$, around each transition guarantees single server semantics [24].

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

$$\overline{M} = \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \int_0^\tau M_\varphi d\varphi$$

where M_φ is the marking at time φ.

Performance Evaluation We will measure the performance of a TMG as the throughput of its transitions. The throughput of a transition t, $\Theta(t)$, is the average number of times t fires per time unit, or cycle time, in the infinitely long execution of the system. Given that we are considering strongly connected TMGs, in the steady state all transitions have exactly the same throughput, Θ. We will describe two well known methods to compute the throughput of a TMG.

Method 1: Each pair $\{a, a^\bullet\}$ of the TMG can be seen as a simple queuing system for which Little's formula [25] can be directly applied. Hence,

$$\overline{M}(a) = \overline{R}(a) \cdot \Theta \tag{2}$$

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 [24]. 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 TMGs 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:

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

The following Linear Programming Problem (LP) includes the constraint (3) for each arc, and the reachability condition for a estimated average marking $\widehat{M}$:

$$\begin{aligned} &\textit{Maximize} \quad \Theta: \\ &\qquad \delta(a^\bullet) \cdot \Theta \leq \widehat{M}(a) \quad \text{for every } a \in A \\ &\qquad \widehat{M} = M_0 + \mathbb{C} \cdot \sigma \\ &\qquad \Theta \leq \min_{t \in T} \frac{1}{\delta(t)} \end{aligned} \tag{4}$$

The last constraint $\Theta \leq \min_{t \in T} 1/\delta(t)$ ensures single server semantics. Such constraint can be dropped if a self-loop arc with one token is introduced around each transition. The solution of LP (4) is the exact throughput of the TMG [24].

Method 2: If C is the set of simple directed cycles in an TMG, its throughput can be determined as [26]:

$$\Theta = \min \left\{ \min_{\mathbf{c} \in C} \frac{M_0(\mathbf{c})}{\sum\limits_{t \in \mathbf{c}} \delta(t)}, \ \min_{t \in T} \frac{1}{\delta(t)} \right\} \tag{5}$$

As in (4), the term $\min_{t \in T} 1/\delta(t)$ enforces single server semantics. Many efficient algorithms for computing the throughput of an TMG exist that do not require an exhaustive enumeration of all cycles [27, 28]. In practice, method 2 usually computes the throughput more efficiently than method 1.

Definition 5 (Critical cycle and arc). *A cycle* $\mathbf{c}$ *satisfying the equality* (5) *is called* critical. *An arc is called critical if it belongs to a critical cycle.*

3.3 Elastic Marked Graphs

Definition 6. *An* Elastic Marked Graph *(EMG) is a tuple* $G = (T, A, M_0, \delta, L)$*, where* (T, A, M_0, δ) *is a TMG and:*

- $\delta : T \to \mathbb{N}^+$ *assigns a positive integer delay to every transition.*
- *For every arc* $a \in A$ *there exists a complementary arc* $a' \in A$ *satisfying the condition* ${}^\bullet a = a'^\bullet$ *and* ${}^\bullet a' = a^\bullet$*. A labelling function* L *maps all arcs of an EMG as forward or backward* $L : A \to \{F, B\}$ *such that* $L(a) = F$ *iff* $L(a') = B$*.*

The delay $\delta(t)$ of a given transition t represents the number of time cycles required by t to perform its computation. Thus, the class of EMGs can model adequately synchronous elastic systems. Typically, in the initial state of an elastic system there is at most one token on a forward arc.

Figure 5 shows an example of EMG. Given that in an EMG every arc a has a complementary arc a', for every pair $\{a, a'\}$, the equality

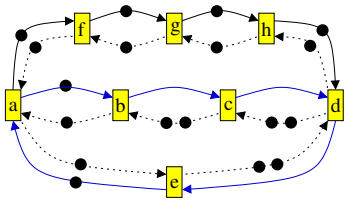


Fig. 5. An example of an elastic marked graph.

$M(a)+M(a')=M_0(a)+M_0(a')=k$ is satisfied, where k is the capacity of the buffer $\{a,a'\}$. Semantically, the pair $\{a,a'\}$ represents the state of an EB. Assume that $L(a)=F$ and $L(a')=B$. We say that the EB is full when $M(a)=k$, $M(a')=0$; when $M(a)=0$, $M(a')=k$ we say that there is a *bubble* in the system. For instance, the EB represented by the arc pair $\{b,c\}$ in Figure 5 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.

4 Slack Matching

The performance of an elastic system may degrade because of unbalanced pipelines. This is a well known problem in asynchronous design. In order to balance the pipelines and improve the performance empty buffers must be added [29–31]. This strategy is known as *slack matching*.

In this section we present two transformations for slack matching of ES: buffer sizing and recycling. The main optimization considered here is *buffer sizing*, that consists on variations of the capacity of the EBs. At the end of the section, we will show that the insertion of bubbles (recycling) also may increase the throughput.

4.1 An Introductory Example

When tokens arrive at the input arcs of a join transition at different times, the early token will stall. The stalled event may generate further stalled events, i.e., it propagates backwards, which may degrade system performance. A very nice explanation of the nature of this phenomenon as well as the exact MILP for slack matching asynchronous design can be found in [31].

Here we try to give an intuitive understanding of the slack matching problem. For this purpose let us simulate the simple EMG depicted in Figure 6. The EMG has the so called unbalanced fork-join structure. The fork transition is a, the join is c. The short branch is $\{a,c\}$ and the long one is $\{a,b,c\}$. All transitions have unit delay. The join transition c is not enabled at time stamp 0. This causes to stall the token on the arc $\{a,c\}$ by one time unit. The rest of the transitions are enabled and will fire. The resulting marking is shown at the configuration in time stamp 1. Now the EB that corresponds to the arc $\{a,c\}$ is full and cannot receive new data. Thus, in time stamp 1 transitions a

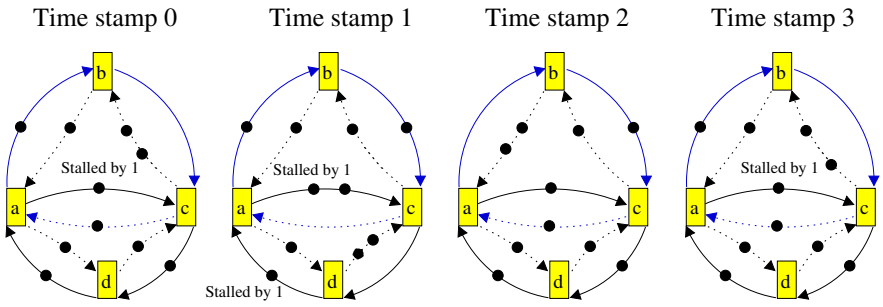


Fig. 6. A stall event backward propagation causes a throughput degradation.

and d are not enabled. This makes the token on the arc $\{d,a\}$ stall by one time unit. At time stamp 2 all transitions are enabled but b. At time stamp 3, the EMG is in the initial state. Each transition has fired twice during three time stamps. Hence, the throughput of the EMG is equal to $\frac{2}{3}$. The critical cycle is $\{a,b,c\}$. It has two tokens and three arcs.

The EMG model allows us to identify when to balance the corresponding ES in order to avoid throughput degradation:

The backward propagation of stalled events leads to the ES throughput degradation iff there are backward arcs on all critical cycles of the corresponding EMG.

In the provided example the critical cycle $\{a,b,c\}$ contains the backward arc $\{c,a\}$.

Buffer sizing and recycling transformations aim to make the throughput independent of backward edges. Hence, the maximum throughput that can be achieved by buffer sizing in a EMG is equal to the throughput of the "forward" TMG that is obtained by removing all backward arcs from the initial EMG.

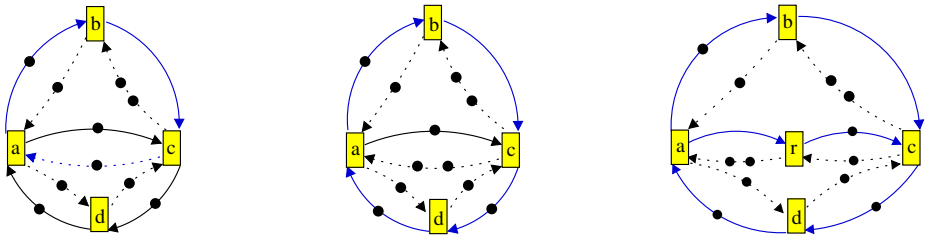
4.2 MILP for Buffer Sizing

Buffer sizing adds tokens to backward arcs, i.e., it increases the capacities of the corresponding EBs.

For example, to remove the backward arc $\{c,a\}$ from the critical cycle in Figure 6 it is enough to increase the capacity of the corresponding EB by one. Figure 7(b) shows the resulting EMG. The throughput is now equal to $\frac{3}{4}$. The critical cycle $\{a,b,c,d\}$ contains only forward arcs. Tokens in the arc $\{d,a\}$ never stall.

The maximum throughput can always be achieved by some proper buffer sizing, however to find a sizing with minimal storage elements overhead is an NP-complete problem [32]; this can be also shown by reducing the feedback arc set problem [33] to minimal buffer sizing.

Let us assume that the throughput of the "forward" TMG is known (it can be computed efficiently with the techniques presented in Section 3.2). Using an estimation of the average marking of the TMG, that was introduced in Section 3.2, one can encode the problem of buffer sizing with minimal storage elements overhead as the following MILP:



(a) (b) (c)

Fig. 7. (a) EMG from the Figure 6, (b) Buffer sizing, (c) Recycling.

$$
\begin{aligned}
\textit{Minimize} \quad & \sum_{a \in A} \Delta M_0(a) : \\
& \widehat{M} = M_0 + \Delta M_0 + \mathbb{C} \cdot \sigma, \\
& \widehat{M}(a) \geq \delta(a^\bullet) \cdot \Theta \qquad \text{for every } a \in A, \\
& \Delta M_0 \in \mathbb{N}^{|A|}.
\end{aligned} \tag{6}
$$

Here Θ is throughput of the corresponding "forward" TMG. For each backward arc a, $\Delta M_0(a)$ contains the number of tokens that need to be added to a in order to reach the throughput Θ. The number $M_0(a) + \Delta M_0(a)$ represents the new marking of a. If a is a forward arc, then $\Delta M_0(a) = 0$ in the solution of (6). In [34], a similar MILP for minimal buffer sizing is presented, which is not based on the MGs theory.

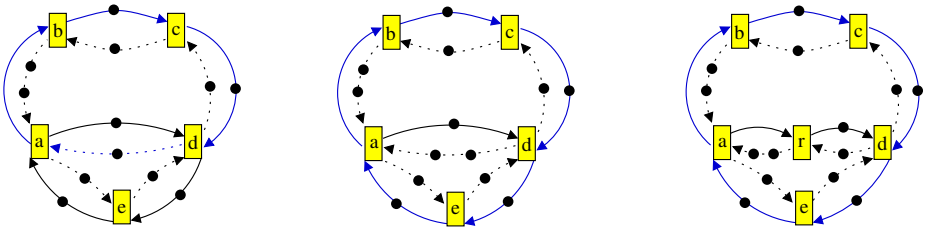
The main disadvantage of buffer sizing is that it increases the complexity and consequently, the area and the combinational delay of the control logic of the ES are increased [35, 36].

4.3 Recycling for Slack Matching

In some situations, the throughput of an ES may be improved by inserting bubbles. Bubble insertion transformation is called *recycling*.

Figure 7(c) shows how the throughput of the EMG depicted in Figure 6 can be increased by inserting the bubble $\{a, r\}$ between transitions a and c. The throughput of the resulting EMG is equal to $\frac{3}{4}$, with critical cycles $\{a, b, c, d\}$ and $\{a, r, c, d\}$.

The main advantage of recycling with respect to buffer sizing is that no extra combinational logic in the control path is required. A weakness is that it may increase the response time of the system. Another drawback is that recycling may not achieve the maximum throughput improvement achieved by buffer sizing. An EMG where this happens is depicted in Figure 8 (a). Assuming unit delays, the throughput in Figure 8(a) is equal to $\frac{3}{4}$, with the critical cycle $\{a, b, c, d\}$. The maximum throughput is given by the "forward" cycle $\{a, b, c, d, e\}$, and it is equal to $\frac{4}{5}$. Applying buffer sizing, this



(a) (b) (b)

Fig. 8. Buffer sizing vs recycling.

throughput can be achieved by increasing the capacity of the buffer $\{a,d\}$, as shown in Figure 8(b).

From the EMG in Figure 8(a), let us try to achieve the same throughput improvement with recycling. Arc $\{d,a\}$ is the only backward arc in the critical cycle. Hence, channel $\{a,d\}$ is the only place where we can insert a bubble to balance the unbalanced fork-join $a-d$. The resulting EMG is depicted in Figure 8(c). It still has throughput $\frac{3}{4}$, due to the new critical cycle $\{a,r,d,e\}$.

In general, the insertion of bubbles in a critical cycle adds a zero-marked arc which may preclude to reach the maximum throughput.

In summary, buffer sizing and recycling are two optimization strategies that can be combined to improve the performance of an ES by removing stall event backward propagation. Depending on the structure of the circuit, buffer sizing can sometimes derive better results, but has a control overhead. For large circuits or circuits containing a regular structure, both transformations will likely lead to the same result.

5 Control Optimization

The main cost of elastizising a synchronous circuit is a control overhead. This section introduces a control simplification technique that reduces this overhead considerably while preserving the performance of the system. For simplicity, in this section we focus on synchronous elastic systems where all transitions have the same delay: $\delta(t_i) = \delta(t_j)$ for every $t_i, t_j \in T$. However, the reader will soon realize that the methods presented in this section can be easily extended to transitions with different delays.

5.1 An Introductory Example

The implementation of an elastic system maps an EMG to an asynchronous or a synchronous control circuit. For instance, Figure 9(a) depicts the elastic circuit corresponding to part (events a, b and f) of the EMG drawn in Figure 10(a). The complexity of the circuit is typically linear in the size of the EMG (e.g., [11]). Therefore, reducing the size of an EMG contributes directly to the size reduction of the control circuit. Based

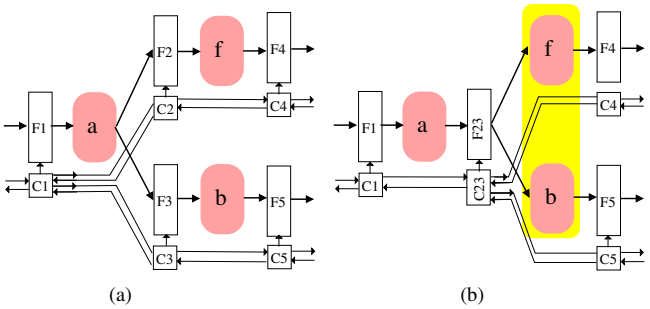


Fig. 9. Sharing a controller and an elastic FIFO.

on this fact, we focus on reducing the number of arcs in an EMG modeling an elastic system as this reduces the number of EBs and the number of channels in the fork and join controllers (that corresponds to transitions with multiple fan-out and fan-in, respectively).

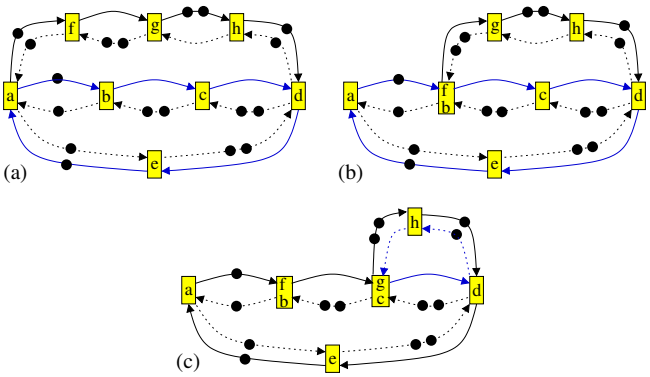


Fig. 10. An example of a elastic marked graph (a), merging b and f (b), and c and g (c).

For example, Figure 9(b) corresponds to the sharing of transitions b and f from the EMG of Figure 10 into a single transition. As a result of this sharing the implementation is simplified by removing one controller, one channel (a pair of handshake wires), and one EB, $F3$, in the data-path that is shared with $F2$.

The goal of this section is to identify a class of transformations that reduce the number of controllers while preserving the performance of the system. In particular, the firing of some transitions can be deliberately postponed in order to be synchronized to other transitions, allowing the sharing of their controllers without degrading the performance of the system.

As described in Section 3.2, the performance of an EMG can be measured by its throughput that is defined as the minimal ratio of the number of tokens to the delay

across all simple cycles and can be efficiently computed [37]. Assuming that the delays of all transitions in Figure 10(a) are equal to 1, the critical cycle is $\{a,b,c,d,e\}$ with a throughput $2/5$ (2 tokens on the arcs of the cycle; delay of the cycle is 5 units). Since the initial marking of the arcs between a and b, and the arcs between a and f is the same, it is possible to merge transitions b and f (as shown in Figure 10(b)) without affecting correctness of computation. The throughput of the system is the same $2/5$ and so is the critical cycle $\{a,\{b,f\},c,d,e\}$. Figure 9(b) shows the corresponding implementation, simplified according to the merging of b and f.

Focusing on the new fork transition $\{b,f\}$ we again determine that the initial marking of arc pairs between $\{b,f\}$ and its successors c and g is the same and therefore it is possible to merge transitions c and g (as shown in Figure 10(c)). However, the throughput of the system is degraded to $1/3$, with a new critical cycle $\{\{c,g\},d,h\}$.

5.2 A Sufficient Condition to Compute Mergeable Transitions

In the example above, the initial marking is used to decide whether two transitions can be merged: when the arcs from an adjacent fork transition have the same initial marking, then the transitions can be merged. The remainder of this section will present a strategy that uses a different marking (called *tight*) to compute mergeable transitions. A tight marking can be considered a variation of the average marking. It better exploits the flexibility of the system, in order to make the markings on the arcs as much as possible equal to maximize the sharing of controllers.

Formally, those pair of transitions that can be merged without degrading the performance of the system are defined:

Definition 7. *Transitions t_i and t_j are said to be* 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 formal definition of tight marking is the following:

Definition 8. *A marking $\tilde{M}$ 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.

Therefore a tight marking satisfies the state equation (condition (7)) and no arc has marking less than $\delta(a^\bullet) \cdot \Theta$ (condition (8)). These two conditions are also satisfied by the average marking, and their relation with the system throughput has been described in Section 3.2. Additionally, the tight marking requires that every transition must have at least one tight incoming arc.

Let us consider the EMG in Figure 11. It has a single critical cycle $\{a,b,c,d,e,f\}$ with a throughput 0.5. Each arc in Figure 11 is labeled with one number if its average and tight markings coincide. When they are different the average marking is listed first

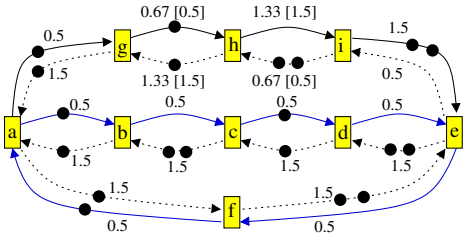


Fig. 11. An EMG illustrating a tight marking.

and the tight marking is shown in square brackets. If the initial marking or the average marking are used, the only mergeable transitions are g and b (g and b are the only two transitions connected to the fork transition a such that $\overline{M}(ag) = \overline{M}(ab)$). However, if the tight marking is used instead, transitions h and c can be additionally merged.

The following theorem formalizes the above concepts on the tight marking [38]:

Theorem 1. *Let $\tilde{M}$ be a tight marking of an EMG G. Transitions t_i and t_j of G 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).*

The first two conditions of Theorem 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 $M(a_i) = M(a_j)$. Such an initialization corresponds to changing the initial marking of the EMG and can be acceptable in many, but not all, applications. 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.

A tight marking can be computed efficiently, as the following proposition states [38]:

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

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

where t_a is a transition that belongs to a critical cycle and k is any real number. The last constraint guarantees the boundedness of the solution. Since the objective function $\Sigma\sigma$ is maximized, the obtained $\tilde{M}$ satisfies that for every transition t there exists an arc $a \in {}^\bullet t$ such that $\delta(t) \cdot \Theta = \tilde{M}(a)$.

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 [39].

The overall strategy for reducing an EMG involves the following steps:

1) Computation of the throughput of the system
2) Computation of a tight marking
3) Determine the sets of mergeable transitions by traversing the tight subgraph
4) Fire transitions to obtain the same marking in the input arcs of the mergeable transitions
5) Merge mergeable transitions and identical arcs

6 Early Evaluation

In an early evaluation setting, operations can execute when enough information at the inputs has been received to determine the value at the outputs. The performance of elastic systems can be enhanced by using early evaluation. This section proposes an analytical model to estimate the performance of an early evaluated marked graph (see [40] for a preliminary work).

6.1 Motivation and Examples

The requirement that all input data must be available to compute a result is too strict in some cases. For example, if a functional unit computes $a = b * c$, it is not necessary to wait for both operands if one of them is already available and known to be zero. Therefore, the result $a = 0$ could be produced by an *early evaluation* of the expression. Early evaluation has been proposed and used in asynchronous design [41, 42].

Usual Petri nets are not capable of modeling early evaluation, since the enabling of transitions is based on AND-causality, i.e., all input conditions must be asserted. Causal Logic Nets from [43] extend Petri nets to allow transition enabling triggered by arbitrary logic guards associated with transitions. This section presents a new model of nets, called *multi-guarded nets* (GN), with the power of modeling early evaluation that associate with a single transition multiple logic guards selected non-deterministically. This non-deterministic selection models interaction of the control with conditions in the data-path.

Figure 12(a) illustrates the usual firing rule in Petri nets (AND-causality). Early evaluation is modeled by *multi-guarded* transitions. A guard is a subset of arcs that can enable a transition. A multi-guarded transition has a set of guards from which one of them is chosen nondeterministically at each firing. Assume that the guards of the transition in Figure 12(b) are $\{\{a\},\{b\},\{c\}\}$, and $\{c\}$ is the guard selected for the next firing. Given that arc c is positively marked, the transition is enabled and will fire. The firing of an early-enabled transition removes a token from every input arc. If an input arc is not positively marked, a *negative token* (-1) is placed in it. This negative token will be cancelled out when a *positive token* arrives at the arc.

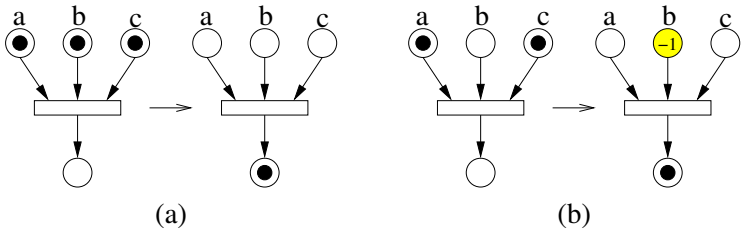
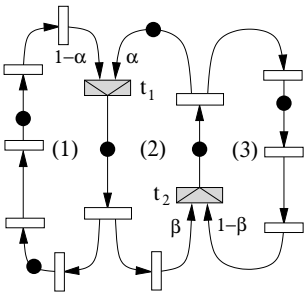


Fig. 12. Multi-guarded transitions: (a) AND-causality; (b) early firing with guard $\{c\}$.

Example 1. The most relevant example of a unit with early evaluation is the multiplexor: the output can be determined as soon as the information of the selected channel arrives, without waiting for the other channels.



β \ α	0.02	0.2	0.4	0.6	0.8	0.98
0.02	0.403	0.403	0.403	0.403	0.403	0.403
0.2	0.423	0.423	0.424	0.424	0.425	0.426
0.4	0.429	0.436	0.442	0.447	0.451	0.453
0.6	0.430	0.441	0.454	0.465	0.480	0.487
0.8	0.430	0.443	0.460	0.479	0.507	0.530
0.98	0.430	0.443	0.461	0.488	0.527	0.584

Fig. 13. Throughput of a GN with probabilistic guards.

Figure 13 depicts a marked graph with three cycles. The shadowed transitions t_1 and t_2 model two multiplexors. Their control signals are assumed not to be critical and are not depicted in the graph. Thus, the two input arcs of the multiplexors model the two input data. Associated to each input arc there is a guard and a real number in the interval $[0,1]$ that indicates the probability for the guard to be selected. Each transition is assumed to have unit delay.

Under a pure Petri net model with AND-causality, the performance of the system would be determined by the most stringent cycle. The throughput Θ_i (tokens/transitions) for each cycle is the following:

$$\Theta_1 = \frac{3}{7} = 0.429 \qquad \Theta_2 = \frac{3}{5} = 0.6 \qquad \Theta_3 = \frac{2}{5} = 0.4$$

Hence, the global throughput of the system would be 2/5. By incorporating early evaluation, the throughput can be increased, as shown in the table at the right-hand side of the figure. When β is close to 0, the system throughput tends to 0.4, i.e., it is almost completely determined by cycle (3). On the other hand, as α and β approach 1, the throughput increases and tends to 0.6, i.e., cycle (2) determines the system throughput.

In general, the throughput lies between 0.4 and 0.6 depending on the probabilities at each multiplexor.

One could think of computing the throughput of the early evaluation system as a weighted sum of the throughputs of the individual loops, i.e., for the above example such a sum would be $\alpha \cdot \beta \cdot \frac{3}{5} + (1-\alpha) \cdot \beta \cdot \frac{3}{7} + \alpha \cdot (1-\beta) \cdot \frac{2}{5} + (1-\alpha) \cdot (1-\beta) \cdot \frac{2}{5}$. Nevertheless, this method is incorrect, since loops may affect each other in a complex interplay.

6.2 Approximate Models for Early Evaluation

Definition 9. *A* Timed Multi-Guarded Marked Graph *(TGMG) is a tuple* $N = \langle T, A, M_0, \delta, H, \alpha \rangle$ *where:*

- $\langle T, A, M_0, \delta \rangle$ *is a timed marked graph TMG working under single server semantics.*
- $H : T \to 2^{2^A}$ *assigns a set of guards to every transition, such that the following condition is satisfied: Every transition t is assigned a set of guards $H(t)$, where every guard $g_i \in H(t)$ is a subset of the input arcs of t, i.e., $g_i \subseteq {}^\bullet t$, and* $\bigcup_{g \in H(t)} g = {}^\bullet t$.
- α *is a function that assigns a strictly positive probability to each guard such that for every guarded transition t:* $\sum_{g \in H(t)} \alpha(g) = 1$.

A guard $g \in H(t)$ is selected first in the initial marking, M_0, and then after each firing of t. The probability of selecting the guard $g \in H(t)$ is $\alpha(g)$. The selected guard of a transition t is *persistent*, i.e., it never changes between the firings of t. If the guard $g \in H(t)$ has been selected for the next firing of t, then t becomes enabled when every arc $a \in g$ has a token ($M(a) > 0$). If t is enabled, it fires $\delta(t)$ time units after becoming enabled. As in conventional transitions, the firing of t removes one token from every input place, and produces one token in every output place. A classical Petri net is simply a GN in which $H(t) = \{{}^\bullet t\}$, for every $t \in T$. Such transitions will be called *simple* transitions.

Analysis through Markov Chains Due to the stochastic nature of selecting guards a TGMG can be viewed as a semi-Markov process [44]. In such a process, the sojourn time of a given state is the elapsed time between the arrival time to the state and the firing time of a transition from such a state.

Figure 14 shows a TGMG (delays of all transitions assumed to be 1) and the associated transition graph of the semi-Markov process. Each arc of the graph corresponds to one time unit. For simplicity, the transitions of the TGMG are named a, b, c, d and arcs ab, ba, ac, cd, da using pairs of names of preset-postset transitions. The states of this graph S_1, S_2, S_3 are the reachable markings. The matrix-like shape depicted at each state correspond to the marking at each state (see graphical explanation in Figure 14). Arcs are labeled with probabilities to be taken (omitted if probability is 1) and a set of firing transitions.

The average time spent at each state (marking) at the steady state can be obtained by solving the set of linear equations corresponding to the semi-Markov process. Let

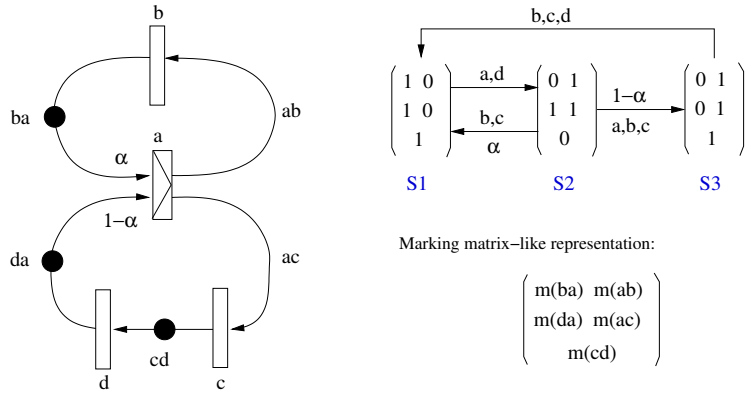


Fig. 14. A TGMG and its associated Markov chain.

Z_1, Z_2, Z_3 be the probabilities to be in the corresponding states S_1, S_2, S_3 in steady state. One can write a set of equations corresponding to the transitions of the process:

$$\begin{aligned} Z_2 &= Z_1 \\ Z_3 &= (1-\alpha)\cdot Z_2 \\ Z_1 + Z_2 + Z_3 &= 1 \end{aligned}$$

The solution is:

$$Z_1 = Z_2 = \frac{1}{3-\alpha} \ , Z_3 = \frac{1-\alpha}{3-\alpha}$$

Transition a is fired with probability 1 from S_1 and with probability $1-\alpha$ from S_2. Therefore, the steady state throughput of transition a is:

$$\Theta(a) = Z_1 + (1-\alpha)\cdot Z_2 = \frac{2-\alpha}{3-\alpha}$$

As in classical TMG s, the steady state throughput of a TGMG is the same for every transition [40], i.e., $\Theta(a) = \Theta(b) = \Theta(c) = \Theta(d)$.

6.3 Performance Estimation

The use of Markov chains allows one to compute the exact throughput of any bounded TGMG. However, it requires an exhaustive exploration of the reachability graph that is exponentially larger than the size of the bounded TGMG. This subsection presents a method to obtain an upper throughput bound via Linear Programming, i.e., the method has polynomial complexity.

For the sake of clarity, we will assume that every transition t has singleton guards, i.e., $|g| = 1$ for every $g \in H(t)$, or is simple, i.e., $H(t) = \{^\bullet t\}$. The set of transitions

with singleton guards is denoted as T_1, and the set of simple transitions is denoted as T_2. Transitions with only one input arc can be included either in T_1 or in T_2. This assumption does not involve a loss of generality: a transition with non-singleton guards can be transformed to a transition with singleton guards with identical behavior [40].

Let $t \in T_2$. As explained in Subsection 3.2, equation (3) is satisfied by each pair $\{a,t\}$ where $a^\bullet = t$. In other words, equation (3) expresses linear relationships between the throughput of a simple transition and the average marking of its input arcs.

For each transition $t \in T_1$, it is also possible to establish a linear relationship between its throughput and the average marking of its input arcs. Let $t \in T_1$ and $prob(enab(t))$ be the probability of t to be enabled in steady state. In other words, $prob(enab(t))$ is the time ratio during which t is enabled. Since transitions have deterministic delays and operate under the single server semantics, the enabling operational law [45] for t is:

$$\delta(t) \cdot \Theta(t) = prob(enab(t)) \qquad \text{for any } t \in T \tag{12}$$

After a number of algebraic manipulations, the value $prob(enab(t))$ can be expressed in terms of the marking of the input arcs of t. In particular, a useful expression is given by Theorem 2 [40]:

Theorem 2. *Let t be a transition with singleton guards, then:*

$$\delta(t) \cdot \Theta(t) = \sum_{a \in {}^\bullet t} \alpha(\{a\}) \cdot \left(\overline{M}(a) - \sum_{i=2}^{\infty} (i-1) \cdot prob(M(a) = i) \right)$$

Corollary 1. *Let t be a transition with singleton guards. If the marking of its input arcs is 1-upperbounded then:*

$$\delta(t) \cdot \Theta(t) = \sum_{a \in {}^\bullet t} \alpha(\{a\}) \cdot \overline{M}(a)$$

else:

$$\delta(t) \cdot \Theta(t) < \sum_{a \in {}^\bullet t} \alpha(\{a\}) \cdot \overline{M}(a)$$

One can combine the constraints in Corollary 1 for transitions in T_1 and the constraint (3) for transitions in T_2 to build a Linear Programming Problem (LP) that maximizes a parameter ϕ, corresponding to the TGMG throughput. One scalar variable suffices since the throughput of all transitions is the same. The resulting LP can be expressed as:

$$\begin{aligned} &\textit{Maximize} \quad \phi : \\ &\quad \delta(t) \cdot \phi \leq \sum_{a \in {}^\bullet t} \alpha(\{a\}) \cdot \widehat{M}(a) \quad \text{for every } t \in T_1 \\ &\quad \delta(t) \cdot \phi \leq \widehat{M}(a) \quad \text{for every } a \in {}^\bullet T_2 \\ &\quad \widehat{M} = M_0 + \mathbb{C} \cdot \sigma \\ &\quad \phi \leq \min_{t \in T} 1/\delta(t) \end{aligned} \tag{13}$$

The vector σ represents the firing count vector that drives the system from the initial marking, M_0, to the estimated average marking $\widehat{M}$. The constraint $\sigma \geq 0$ has been dropped since for any non-positive σ, a positive σ exists that delivers the same maximum value of ϕ (this is due to the fact that C is not a full rank matrix). The last constraint $\phi \leq \min_{t \in T} 1/\delta(t)$ guarantees single server semantics.

The LP (13) always has solution since all its constraints must hold in the steady state. Given that the throughput variable, ϕ, is maximized, the obtained value is an upper throughput bound [40].

Theorem 3. *Let N be a* TGMG*. The solution of* (13) *gives an upper bound for the steady state throughput of the* TGMG*.*

Example 2. Consider again the 1-bounded TGMG from Figure 14 (delays of all transitions assumed to be 1). The associated LP problem is:

$$
\begin{array}{ll}
\text{Maximize} \quad \phi: & \\
\phi \leq \overline{M}(ab) & \text{for transition } b \\
\phi \leq \overline{M}(ac) & \text{for transition } c \\
\phi \leq \overline{M}(cd) & \text{for transition } d \\
\phi \leq \alpha \cdot \overline{M}(ba) + (1-\alpha) \cdot \overline{M}(da) & \text{for transition } a \\
ba = 1 + \overline{M}(b) - \overline{M}(a) & \text{for arc } ba \\
da = 1 + \overline{M}(d) - \overline{M}(a) & \text{for arc } da \\
ab = \overline{M}(a) - \overline{M}(b) & \text{for arc } ab \\
ac = \overline{M}(a) - \overline{M}(c) & \text{for arc } ac \\
cd = 1 + \overline{M}(c) - \overline{M}(d) & \text{for arc } cd
\end{array}
$$

The solution to this problem is

$$\phi = \frac{2-\alpha}{3-\alpha}$$

which, in this case, corresponds exactly to the solution we have obtained with Markov chain analysis.

7 Retiming and Recycling

In this section we will show how a well-known optimization technique (retiming [46]) can be combined with the insertion of empty buffers (recycling) for performance optimization. The EMG representation does not capture information about the combinational delay of a node, so in this section our representation of an elastic system is based on the *retiming graph* of Leiserson and Saxe [46].

7.1 An Introductory Example

In logic synthesis the usual representation of a synchronous sequential circuit is a set of combinational blocks interconnected via memory elements (registers). Figure 15(a)

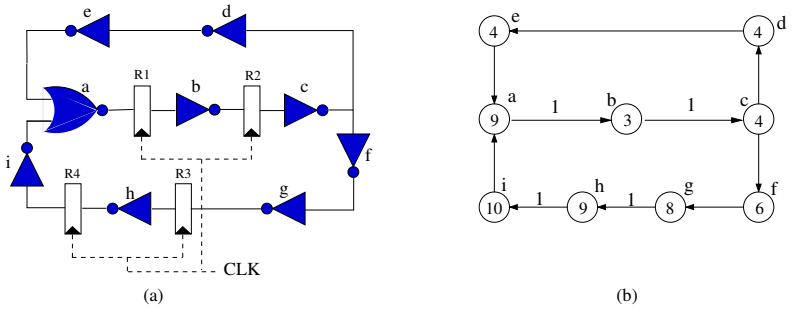


Fig. 15. (a) A synchronous sequential circuit, (b) its retiming graph.

gives an example of a simple sequential circuit. This circuit has nine simple combinational blocks, denoted with lower case letters $a, b, \ldots, i$ and four registers R_1, R_2, R_3, R_4. Every gate computes some boolean function. The gate a is a usual representation of a NOR-gate, which implements a boolean function of two variables $f(x_1, x_2) = \overline{x_1 \vee x_2}$. The *delay* of the gate is the amount of time required to recompute the output value when some inputs are changed. A *combinational path* is a sequence of directly connected gates, i.e., without registers along the sequence. The sequence c, d, e, a from Figure 15 is a combinational path, while the sequence a, b is not. The combinational path delay is the sum of the delays of its nodes. In order to have a well-defined physical design, combinational paths must not form cycles, i.e., each cycle must have at least one register.

Every time that the global clock signal (denoted as CLK on the figure) arrives, the registers become "transparent", i.e., the input data becomes output data. For example, the result computed by gate b during the previous clock cycle becomes the input for gate c. The amount of time between two consecutive clock signals (a *clock period*) should allow each gate to recompute its output value. In order to guarantee a correct functionality, the maximum combinational path delay of the circuit, which is called *cycle time*, should be less than or equal to the clock period.

The retiming technique represents sequential circuits as weighted directed graphs (retiming graph). The nodes of the graph model gates. Each node is labeled with its delay. A directed edge of the graph models an interconnection between gates, and is weighted with a register count. The register count is the number of registers along the connection. Figure 15(b) shows the corresponding retiming graph for the sequential circuit depicted on the Figure 15(a). The nodes (circles) are labeled with their delays. The edges are labeled with the corresponding register number, unlabeled edges have no registers. The delay of the combinational path c, f, g is equal to 18 time units, for path c, d, e, a it is equal to 21. The cycle time of the circuit with delays is equal to 21.

A retiming r on a given graph assigns an integer to each node of the graph. This assignment transforms the edge register count as follows: Assume that edge e has source node u, target node v and register count $w(e)$. Then, after applying retiming r, e will have register count $w'(e) = w(e) + r(v) - r(u)$. Let us exemplify this technique by applying the following retiming on the retiming graph in Figure 16(a): $r(a) = 1, r(c) = -1$, r is

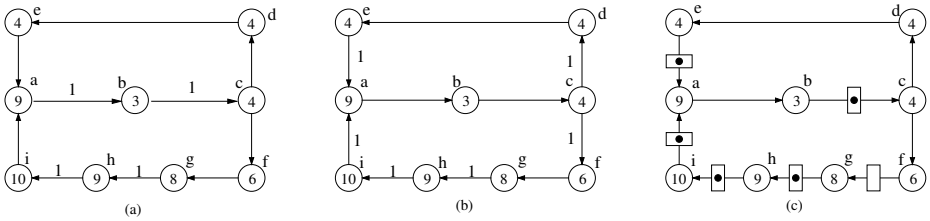


Fig. 16. (a) Retiming graph, (b) Min-delay retiming configuration, (c) Retiming and recycling.

zero for the rest of the nodes. In order to apply $r(a) = 1$ it is enough to remove one register from each output edge of node a and add one register to each input edge of a; $r(c) = -1$ moves register across node c in opposite direction. Figure 16(b) shows the resulting graph.

Retiming may change the cycle time and the number of registers in the circuit while preserving its sequential behavior. In the graph in the Figure 16(b) the combinational path with the greatest delay is a, b, c. Then, the cycle time is equal to $9 + 3 + 4 = 16$ time units.

In order to describe an ES, we should be able to distinguish registers that contain valid data (dots) from empty registers (bubbles). For this purpose, we will add a new edge label into the retiming graph. This label represents the total number of registers (dots and bubbles) on the edge. The register count of the retiming graph now represents only the number of dots on the edge. Figure 16(c) shows an example. The empty boxes represent registers with non-valid data (register count=1, dot count=0), boxes with dots represent registers with valid data (register count=dot count=1). There are two combinational paths that determine the cycle time: a, b and c, d, e. The cycle time is equal to 12 time units.

The bottom directed cycle has 4 dots and 5 registers. Therefore, it does not produce valid data every clock cycle, but 4 valid data every 5 cycles, i.e., its *processing rate* is $\frac{4}{5}$. The *effective cycle time* is given by its cycle time divided by its processing rate, this yields $12 \cdot \frac{5}{4} = 15$. This effective cycle time is better than the one of the original non-elastic circuit which was 16 (the processing rate of a system without bubbles is equal to one).

In this section we show how the *minimal effective cycle time* of an ES represented as a retiming graph can be found. An exact solution of the retiming and recycling problem is specified with MILPs.

7.2 Marked Graphs and Retiming

The retiming graph (RG) is isomorphic to a TMG. Each combinational block corresponds to a node, each connection corresponds to an edge. The registers in the retiming graph are represented by tokens in the MG. This way, the firing rules of a MG coincide with the backward retiming rule: each time a node is retimed, registers are removed from the input edges and added to the output edges.

Definition 10 (Retiming Graph). *A* Retiming Graph *(RG) is a TMG* (N, E, R_0, δ). R_0 *represents an initial assignment of registers with informative data (dots) to the edges of the graph.*

In Section 3.1 basic structural properties of a MG were introduced. The retiming interpretation of these properties is the following:

Retiming interpretation of liveness (Property 1, Section 3.1): *every cycle should have at least one register to avoid combinational cycles in the circuit netlist.*

Retiming interpretation of token preservation (Property 3, Section 3.1): This property has two directions. The $\Rightarrow$ direction corresponds to a well-known result in retiming: *A valid retiming preserves the number of registers at each cycle.* The important direction is $\Leftarrow$ that provides a new result for the theory of retiming [47]: *If an assignment of registers has the same number of registers at each cycle as the initial circuit, then the assignment is a valid retiming.*

Thus, we can reduce the retiming problem to a reachability problem in MGs.

In order to represent bubbles we associate another register assignment with a RG.

Definition 11 (Retiming and recycling configuration of a RG). *A retiming and recycling configuration (R&R) of a RG is a register assignment* $R : E \rightarrow N$.

An important question is: what is a valid R&R? The answer is easy: let us take any valid retiming configuration of the RG and let us add any arbitrary number of registers (bubbles) to every edge. That is, set register count R as follows: $R(e) = R_0(e) + k,\ k \in \mathbb{N}$. The resulting R&R is valid. Therefore, any integer vector R that satisfies to the following inequalities:

$$R \geq \widehat{R} = R_0 + \mathbb{C} \cdot \sigma \geq 0,\ \ R, \widehat{R} \in \mathbb{N}^{|E|} \tag{14}$$

is a valid R&R. In (14), $\mathbb{C}$ is the incidence matrix of the RG, $\widehat{R}$ represents the *retiming subset* of the solution (the registers containing only dots), and R represents registers containing dots and bubbles.

A bubble in a valid R&R is represented as follows: $R(e) = 1, \widehat{R}(e) = 0$, e.g., edge (f, g) in Figure 16(c). If the edge e has two registers and only one dot then it has the followings register counts: $R(e) = 2, \widehat{R}(e) = 1$. The register counts of an edge without registers are $R(e) = 0, \widehat{R}(e) = 0$. The difference between both vectors, $R - \widehat{R}$, represents the vector of registers containing the bubbles introduced by recycling.

Let $\tau(R)$ be the cycle time of R, i.e., the greatest delay of the path without registers in the RG. For instance, the cycle time of the R&R in Figure 16(c) is equal to 12. Let $\Theta(R)$ be the throughput of R, i.e., the minimal dots to registers ratio of all directed cycles of the RG[3]. The cycle ratios for the top and bottom cycles of Figure 16(c) are 1 and 4/5, respectively. Therefore, $\Theta(R) = 4/5$. The main performance measure of R is the *effective cycle time*. The effective cycle time of a R (ξ(R)) is the ratio of its cycle time and the throughput.

Now we give an overview of the strategy to find, for a given RG, a R with the minimal effective cycle time. The reader can refer to [47] for details.

[3] This throughput model assumes that backward arcs of the corresponding EMG do not constraint the throughput. This always can be achieved by proper buffer sizing (see Section 4).

7.3 Basic MILPs for Retiming and Recycling

Given a cycle time τ_c and a throughput $\Theta_c, 0 < \Theta_c \leq 1$. A registers assignment R is a valid R&R with $\tau(R) \leq \tau_c$ and $\Theta(R) \geq \Theta_c$ if it satisfies the following three sets of inequalities:

$$\texttt{RR}(\tau_c, \Theta_c) \equiv \begin{cases} R \geq R_0 + \mathbb{C} \cdot \sigma_1 \geq 0, \\ R \cdot \Theta_c \leq R_0 + \mathbb{C} \cdot \sigma_2, \\ \texttt{Path\_Constraints}(R, \tau_c), \\ R \in \mathbb{N}^{|E|}, \sigma_1 \in \mathbb{Z}^{|N|}. \end{cases} \tag{15}$$

The first set of the inequalities guarantees that R is valid (see (14)). The second set of the inequalities guarantees that the throughput of R is at least equal to Θ_c. They can be derived using MG performance theory [24] or the linear programming formulation of the minimal cycle ratio problem [28]. The $\texttt{Path\_Constraints}(R, \tau_c)$ is a set of linear inequalities that guarantees the delay of all combinational paths is at most τ_c [47].

7.4 Minimal Effective Cycle Time

Among all R&R configurations that satisfy constraints (15), the ones with minimal cycle time can be found with the following MILP:

$$\begin{aligned} &\textit{Minimize} \quad \tau : \\ &\quad \text{subject to } \texttt{RR}(\tau, \Theta_c). \end{aligned} \tag{16}$$

Similarly, the throughput can be maximized (cycle time τ_c is constant):

$$\begin{aligned} &\textit{Maximize} \quad \Theta : \\ &\quad \text{subject to } \texttt{RR}(\tau_c, \Theta). \end{aligned} \tag{17}$$

Problem (17) with Θ being a variable is neither linear nor convex. However, the throughput constraints in (15) can be modified as follows:

$$\frac{1}{\Theta} \cdot R_0 \geq R + \mathbb{C} \cdot \sigma_2'.$$

Then, after substituting $x = \frac{1}{\Theta}$, the throughput can be maximized with the following MILP:

$$\begin{aligned} &\textit{Minimize} \quad x : \\ &\qquad R \geq R_0 + \mathbb{C} \cdot \sigma_1 \geq 0, \\ &\qquad R_0 \cdot x \geq R + \mathbb{C} \cdot \sigma_2, \\ &\qquad \texttt{Path\_Constraints}(R, \tau_c), \\ &\qquad R \in \mathbb{N}^{|E|}, \sigma_1 \in \mathbb{Z}^{|N|}. \end{aligned} \tag{18}$$

Let $R(\tau, \Theta)$ be a R&R with cycle time τ and throughput Θ. We say that $R_1(\tau_1, \Theta_1)$ is dominated by $R_2(\tau_2, \Theta_2)$ iff $\Theta_1 = \Theta_2$ and $\tau_2 < \tau_1$. If R_1 is dominated by R_2 then $\xi(R_1) > \xi(R_2)$ and R_1 cannot provide the minimal effective cycle time. We say that $R(\tau, \Theta)$ is non-dominated if it is not dominated by any another configuration. Using MILPs (16) and (18) we can find all non-dominated R&R configurations and consequently the minimal effective cycle time.

8 Conclusions and Open Problems

When the behavior derived from the structure of a circuit is modeled at a low level of granularity, concepts like concurrency and elasticity appear in a natural way. The analysis of such systems can take advantage of the strong analogy between the structure and the behavior of a circuit and the structure and token flow of a Petri net.

This paper has reviewed several problems of elastic circuits that can be abstracted and reduced to problems in Petri nets, mainly marked graphs. The variability of computation and communication latencies and the increasing demand in relaxing the strong requirements imposed by global clocks open the door to new design paradigms with more complex models.

This is an area in which the sinergism between two worlds can be exploited. The existing knowledge in the theory of Petri nets can be effectively used to model and reason about problems that are actually emerging in the area of digital circuit design.

Open Problems Several extensions of the models used in this paper can lead to a more accurate description of an elastic system. Two main extensions on the model might be considered for this purpose: (a) introduce early evaluated nodes in the problems considered in Sections 4, 5 and 7, and (b) incorporate variable latencies in the nodes of the EMG.

An EMG extended with variable latencies on the nodes captures the variability of some nodes, by associating a probability function to the delays of the transitions. The methods proposed in this paper must be revised to handle these extensions.

References

1. Sutherland, I. E. : Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.
2. Muller, D. E., Bartky, W. S. : A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243. Harvard University Press, April 1959.
3. Martin, A. J. : Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.
4. Sparsø, J., Furber, S., editor. *: Principles of asynchronous circuit design: a systems perspective*. Kluwer Academic Publishers, 2001.
5. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A. *: Logic synthesis of asynchronous controllers and interfaces*. Springer-Verlag, 2002.
6. O'Leary, J., Brown, G. : Synchronous emulation of asynchronous circuits. *IEEE Transactions on Computer-Aided Design*, 16(2):205–209, February 1997.
7. Peeters, A., van Berkel, K. : Synchronous handshake circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 86–95. IEEE Computer Society Press, March 2001.
8. Carloni, L.P., McMillan, K.L., Sangiovanni-Vincentelli, A.L. : Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design*, 20(9):1059–1076, September 2001.
9. Carloni, L.P., Sangiovanni-Vincentelli, A.L. : Coping with latency in SoC design. *IEEE Micro, Special Issue on Systems on Chip*, 22(5):12, October 2002.
10. Jacobson, H.M., Kudva, P.N.,Bose, P., Cook, P.W., Schuster, S.E., Mercer, E.G., Myers, C.J. : Synchronous interlocked pipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 3–12, April 2002.

11. Cortadella, J., Kishinevsky, M., Grundmann, B. : Synthesis of synchronous elastic architectures. In *Proc. ACM/IEEE Design Automation Conference*, pages 657–662, July 2006.
12. Cortadella, J., Kishinevsky, M. : Synchronous elastic circuits with early evaluation and token counterflow. In *Proc. ACM/IEEE Design Automation Conference*, pages 416–419, June 2007.
13. Dennis, J.B. : Modular asynchronous control structures for a high performance processor. In *Project MAC Conf. on Concurrent Systems and Parallel Computation*, pages 55–80, 1970.
14. Dennis, J.B., Patil, S.S. : Speed-independent asynchronous circuits. In *Proc. Hawaii International Conf. System Sciences*, pages 55–58, 1971.
15. Misunas, D. : Petri nets and speed independent design. *Communications of the ACM*, 16(8):474–481, August 1973.
16. Linder, D.H., Harden, J.C. : Phased logic: Supporting the synchronous design paradigm with delay-insensitive circuitry. *IEEE Transactions on Computers*, 45(9):1031–1044, September 1996.
17. Cortadella, J., Kondratyev, A., Lavagno, L., Sotiriou, C. : Desynchronization: Synthesis of asynchronous circuits from synchronous specifications. *IEEE Transactions on Computer-Aided Design*, 25(10):1904–1921, October 2006.
18. Rosenblum, L.Y., Yakovlev, A.V. : Signal graphs: from self-timed to timed ones. In *Proceedings of International Workshop on Timed Petri Nets*, pages 199–207, Torino, Italy, July 1985. IEEE Computer Society Press.
19. Yoeli, M. : Specification and verification of asynchronous circuits using marked graphs. In K. Voss, H. J. Genrich, and G. Rozenberg, editors, *Concurrency and Nets, Advances in Petri Nets*, pages 605–622. Springer-Verlag, 1987.
20. Yakovlev, A., Gomes, L., Lavagno, L., editor. *: Hardware Design And Petri Nets*. Kluwer Academic Publishers, 2000.
21. Schrijver, A. *: Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.
22. CPLEX. `http://www.ilog.com/products/cplex`.
23. Murata, T. : Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, April 1989.
24. Campos, J., Silva, M. : Structural techniques and performance bounds of stochastic Petri net models. In *Advances in Petri Nets 1992*, volume 609 of *LNCS*. Springer, 1992.
25. Little, J.D.C. : A proof of the queueing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.
26. Ramamoorthy, C.V., Ho, G.S. : Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Trans. Software Eng.*, 6(5):440–449, 1980.
27. Karp, R. : A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.
28. Dasdan, A., Irani, S.S., Gupta, R.K. : Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In *Proc. 36th Design Automation Conference*, pages 37–42, 1999.
29. Williams, T.E. : Performance of iterative computation in self-timed rings. *Journal of VLSI Signal Processing*, 7(1/2):17–31, February 1994.
30. Manohar, R., Martin, A.J. : Slack elasticity in concurrent computing. In J. Jeuring, editor, *Proc. 4th International Conference on the Mathematics of Program Construction*, volume 1422 of *Lecture Notes in Computer Science*, pages 272–285, 1998.
31. Beerel, P.A., Kim, N-H, Lines, A., Davies, M. : Slack matching asynchronous designs. In *Proc. of the 12th Int. Symp. on Asynchronous Circuits and Systems*, 2006.
32. Rodriquez-Beltran, J., Ramirez-Trevino, A. : Minimum initial marking in timed marked graphs. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'2000)*, volume 4, pages 3004–3008, October 2000.

33. Garey, M.R., Johnson, D.S. *: Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
34. Lu, R., Koh, C-K. : Performance optimization of latency insensitive systems through buffer queue sizing of communication channels. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 227–231, 2003.
35. Lu, R., Koh, C-K. : Performance analysis and efficient implementation of latency insensitive systems. ECE Technical Reports, March 2003.
36. Chelcea, T., Nowick, S.M. : Robust interfaces for mixed-timing systems. *IEEE Trans. VLSI Syst.*, 12(8):857–873, 2004.
37. Dasdan, A., Gupta, R.K. : Faster maximum and minimum mean cycle algorithms for system performance analysis. *IEEE Transactions on Computer-Aided Design*, 17(10):889–899, 1998.
38. Carmona, J., Júlvez, J., Cortadella, J., Kishinevsky, M. : Performance-preserving clustering of elastic controllers. Technical Report LSI-08-7-R, Department of Software, Universitat Politècnica de Catalunya, 2007.
39. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E. *: Introduction to Algorithms*. McGraw - Hill Higher Education, 2001.
40. Júlvez, J., Cortadella, J., Kishinevsky, M. :performance analysis of concurrent systems with early evaluation. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, November 2006.
41. Brej, C.F., Garside, J.D. : Early output logic using anti-tokens. In *Int. Workshop on Logic Synthesis*, pages 302–309, May 2003.
42. Reese, R.B., Thornton, M.A., Traver, C., Hemmendinger, D. : Early evaluation for performance enhancement in phased logic. *IEEE Transactions on Computer-Aided Design*, 24(4):532–550, April 2005.
43. Yakovlev, A., Kishinevsky, M., Kondratyev, A., Lavagno, L., Pietkiewicz-Koutny, M. : On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9(3):189–233, 1996.
44. Wolff, R.W. *: Stochastic modeling and the theory of queues*. Prentice Hall, 1989.
45. Chiola, G., Anglano, C., Campos, J., Colom, J.M., Silva, M. : Operational analysis of timed Petri nets and application to the computation of performance bounds. In F. Baccelli, A. Jean-Marie, and I. Mitrani, editors, *Quantitative Methods in Parallel Systems*, pages 161–174. Springer, 1995. Also appears in Procs. PNPM93.
46. Leiserson, C.E., Saxe, J.B. : Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.
47. Bufistov, D., Cortadella, J., Kishinevsky, M., Sapatnekar, S. : A general model for performance optimization of sequential systems. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 362–369, November 2007.