Retiming and recycling for elastic systems with early evaluation*

Dmitry Bufistov Jordi Cortadella Marc Galceran-Oms
Universitat Politècnica de Catalunya
Barcelona, Spain

Jorge Júlvez
Universidad de Zaragoza
Zaragoza, Spain

Mike Kishinevsky
Strategic CAD Lab, Intel Corp
Hillsboro, OR, USA

Abstract

Retiming and recycling are two transformations used to optimize the performance of latency-insensitive (a.k.a. synchronous elastic) systems. This paper presents an approach that combines these two transformations for performance optimization of elastic systems with early evaluation. The method is based on Mixed Integer Linear Programming.

On a set of random benchmarks the proposed method achieves, in average, 14.5% performance improvement over min-delay retiming configurations.

1 Introduction

Latency-insensitive (a.k.a. synchronous elastic) systems tolerate changes in communication and computation latencies [1, 2]. The term "elastic system", ES, will be used in this paper.

An elastic system can be viewed as a composition of combinational blocks and elastic FIFOs connected by channels. A channel is comprised of data wires and a pair of handshake control signals: (valid, stop). The basic case of an elastic FIFO, called elastic buffer, EB, has a latency of one clock cycle and a capacity to store two pieces of information (*tokens*). An EB initially storing one token of information is an elastic equivalent of a synchronous register. An empty EB which contains no tokens is called a *bubble*.

The *valid* and *stop* bits in elastic channels implement a handshake protocol between the sender and the receiver. The valid bit, going in the forward direction, is used by the

*This is an extended version of a paper published at the Design Automation Conference, San Francisco, July 2009.

This work has been supported by FPU grant AP2005-4866, FI grant B1 00063, research projet CICYT TIN2007-66523, Juan de la Cierva fellowship from the Spanish Ministry of Education and Science and partially supported by a grant from Intel Corp., CICYT TIN2004-07925.

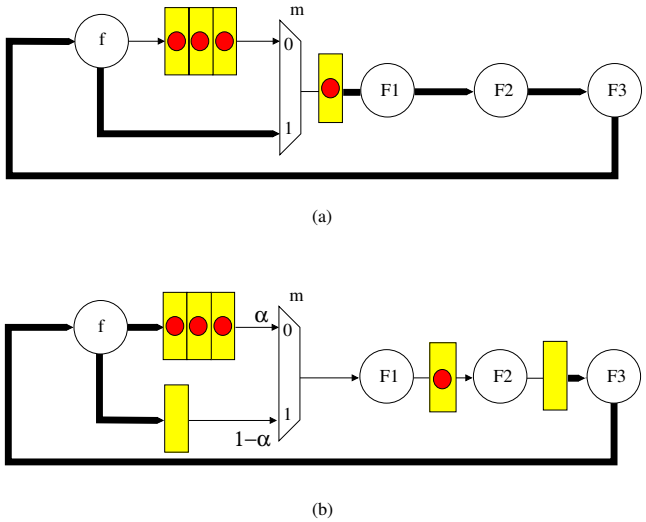

Figure 1: (a) A retiming and recycling graph, (b) Optimal retiming&recycling solution

sender to indicate when useful data is being sent. The stop bit, going in the backward direction, is used for stalling the sender by propagating *back-pressure* when the receiver is not ready.

In ESs, the latency of propagating data and valid bit in the forward direction is typically the same as the latency of propagating stop signals in the backward direction. This makes ESs highly scalable as no global stall needs to be propagated combinationally throughout the whole system.

Any synchronous circuit can be transformed into an equivalent ES following a simple automatic flow [2, 3].

A key aspect of ESs is that they accept a set of valid transformations [4] that preserve the circuit behavior regardless the timing characteristics of its components.

We refer the reader to [1, 2] for basic concepts on ESs.

1.1 Retiming and Recycling Graph

Retiming [5] is a well-known technique for sequential optimization of synchronous circuits. It moves registers across combinational blocks to minimize the clock cycle or area. It preserves the sequential behavior of a circuit. In ESs, retiming moves EBs instead of regular registers.

An ES is modeled by a *Retiming and Recycling Graph* (RRG). Each node of the graph is a combinational block with an associated combinational delay. Each edge represents a connection between combinational blocks labelled with EBs when needed. The RRG can be viewed as an extension of the retiming graph [5].

Figure 1(a) shows an example of an RRG. Only the datapath of the ES is drawn. Each box at the edges represents an EB. If the box is empty, the EB contains no valid information. If the box is marked with a dot, the EB contains one token. E.g., the top

edge between nodes f and m has three EBs each labelled with a token. Multiplexors (such as node m) are drawn by using a different symbol than other nodes. Later we will see why.

Assuming that nodes F1, F2, F3 have unit combinational delay while other nodes have zero delay, the *cycle time* of the RRG in Figure 1(a) is equal to three time units, determined by the *critical combinational path* $F1, F2, F3, f, m$ (marked with the thick line in the figure).

1.2 Retiming and Recycling (RR)

In ESs it is possible to insert an empty EB at any channel of the system preserving sequential behavior with respect to valid tokens of information. Empty buffer insertion is called *recycling* [6].

The critical combinational cycle $F1, F2, F3, f, m$ in Figure 1(a) with a bottom edge f, m contains only one EB. Retiming preserves the total number of EBs at each directed cycle [5]. Thus, the moves of retiming cannot reduce the cycle time in this example: 3 is minimal cycle time achievable by retiming. However, RRG in Figure 1(b) obtained by applying one retiming move and inserting two bubbles has a smaller cycle time of 1 time unit.

This reduction of a cycle time is, however, useless, since the actual performance of the ES has not improved. Indeed, two inserted bubbles reduce the *throughput* of the ES (defined as the amount of useful work done per cycle) to $\frac{1}{3}$[1]. The multiplexor needs to wait for both valid inputs before computing a new token. This is the reason for throughput degradation.

To compare the performance of two ESs the *effective cycle time* metrics is used. The effective cycle time is the ratio of the cycle time to the throughput. Ignoring the delay overhead of inserting extra EBs the effective cycle time of both ESs shown in Figures 1 is the same. It is equal to 3 time units. Minimizing the effective cycle time of an ES is the main goal of this paper.

1.3 Early evaluation

Conventional ESs are based on *late evaluation*: the computation is initiated only when all inputs are available. Sometimes this requirement is too strict. For example, once a multiplexor received a select signal, it is sufficient to wait for the selected data channel to produce a token. The other data channel is a don't care.

Early evaluation takes advantage of this flexibility to improve the performance of the ES. Care must be taken of the late arriving irrelevant tokens to avoid spurious enabling of functional units. The control logic for ESs with early evaluation is presented in [3]. When early evaluation occurs, a negative token, also called *anti-token*, is generated in the late channels that were not using for enabling the block. When an

[1] Elastic FIFOs have fixed sizes. These sizes may affect the throughput of an ES. Thus, in general, tokens to latency ratio provides an upper bound to the actual throughput of the ES. This paper assumes that each elastic FIFO is big enough for storing the tokens it may receive, i.e., the performance of the ES is determined by the forward critical paths, not by the backward ones. It is always possible to optimize FIFO sizes such that the above assumption is true by optimally sizing FIFOs [7].

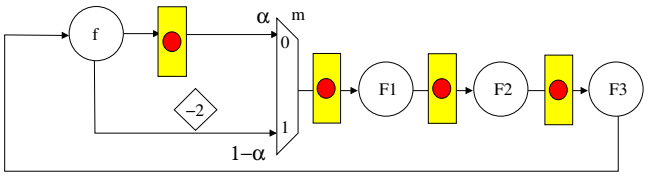

Figure 2: Optimal retiming&recycling solution with early evaluation

anti-token and a token meet in the same channel, they cancel each other. Anti-tokens can be *passive*, waiting for the token to arrive, or *active*, traveling backwards through the control until they meet a token. Elastic FIFOs can store and propagate anti-tokens the same way they store and propagate tokens. Notice that an empty EB is equivalent to an EB containing one token followed by an anti-token ($0 = 1 - 1$). This rule can be often applied to enable retiming of EBs that have been initialized with a different number of tokens.

1.4 Motivational Example

Let us show how RR applied together with early evaluation can improve performance of the RRG from Figure 1. Assume that the select channel of the multiplexor is always valid and it chooses the top data channel with probability $0 < \alpha < 1$, and the bottom channel with probability $(1 - \alpha)$.

The behavior of ESs with early evaluation can be modeled using *Markov chains* [8]. Although this approach does not scale in general (and therefore, faster analytical methods for computing upper bound of the throughput or simulation are used in the proposed method), it can be used for analysis of this small example to compute an exact expression for the throughput. Recall that with late evaluation the effective cycle time of the ES in Figure 1(b) is equal to 3. With early evaluation, the throughput is 0.491 for $\alpha = 0.5$. Hence, the effective cycle time is $1/0.491 \approx 2.037$ time units. For $\alpha = 0.9$ the throughput is higher and is equal to 0.719 and the effective cycle time is approximately 1.39 time units.

Using RR it is possible to further improve the performance in the example. The obtained optimal solution is shown in Figure 2. Resolving the Markov chain for the ES in Figure 2, the following expression for the throughput is obtained: $1/(3 - 2\alpha)$. For $\alpha = 0.9$, the throughput is equal to $\frac{5}{6} \approx 0.833$ that is approximately 16% better than the throughput for the ES from Figure 1(b) with an early evaluation mux.

The bottom channel coming to the multiplexor contains two anti-tokens (drawn in the rhombus). Note that the total sum of tokens is an invariant and is equal to four for the top cycle and to one $(3 - 2)$ for the bottom cycle.

For larger systems, the upper bound on the throughput is obtained by solving a linear programming (LP) model (fast method), or the system is simulated (slower, but more accurate throughput estimation).

Our contribution. A method for minimization of the effective cycle time of ESs with early evaluation is presented. The work is an extension of the paper about per-

formance optimization of ESs with late evaluation [9]. However the extension is non-trivial:

- The performance core of the mathematical programming formulation has to capture early evaluation. There are no known polynomial time algorithms for exact throughput calculation of a such type of systems.
- The proposed solution is a non-convex quadratic mixed integer programming problem. Such problems provide a big challenge for existing solvers. However, the special structure of the problem allows to find some non-dominated points (Pareto frontier). A Pareto point can be found by solving two mixed integer linear programming problems.

Experimental results show that the performance of ESs with early evaluation can be significantly improved. This in contrast with ESs with late evaluation, where recycling can improve retiming only in relatively rare cases of highly unbalanced delays in different paths [9].

2 Background

This section formalizes basic concepts and gives an overview of retiming and recycling (RR).

Definition 2.1 (RRG) *A* Retiming and Recycling Graph *(RRG) is a tuple* $\langle S, \beta, R_0, R, \gamma \rangle$*, where:*

- $S = (N, E)$ *is the underlying multi-graph of the ES,* N *is the set of nodes and* E *is the set of edges. The set* N *is partitioned into* N_1 *and* N_2*:* N_1 *includes the simple combinational nodes and* N_2 *the early evaluation nodes.*
- $\beta : N \rightarrow \mathbb{R}^+$ *assigns a combinational delay to each node.*
- $R_0 : E \rightarrow \mathbb{Z}$ *is the number of the tokens on each edge. If negative,* R_0 *is the number of anti-tokens. To ensure liveness the sum of tokens on each directed cycle of* S *must be positive.*
- $R : E \rightarrow \mathbb{Z}^+$ *is the number of EBs on each edge. Condition* $R \geq R_0$ *must hold.*
- $\gamma : E \rightarrow \mathbb{R}^+ \backslash \{0\}$ *is the branch selection probability for input edges of early evaluation nodes* $n \in N_2$*. The sum of the probabilities for all inputs of an early evaluation node* $n \in N_2$ *is equal to one, i.e.:* $\sum_{e=(n_i,n) \in E} \gamma(e) = 1.$

As an example, the values of R_0, R and γ of the top (bottom) edge (f, m) of the RRG in Figure 1(b) are 3, 3 and α (0, 1 and $1 - \alpha$).

Let us now define some concepts related to timing and performance in RRGs.

Definition 2.2 (Combinational path) *Given an RRG, a* combinational path CP *is a sequence of nodes and edges* $n_0 \xrightarrow{e_1} n_1 \xrightarrow{e_2} \cdots \xrightarrow{e_k} n_k$ *such that* $R(e_i) = 0$ *for all edges in the path. The delay of the combinational path* CP *is equal to* $\sum_{n_i \in CP} \beta(n_i)$*.*

For example, the path formed by the nodes $F1, F2, F3$ in the figure 1(a) is a combinational while the path $f, m, F1$ is not.

Definition 2.3 (Cycle time) *The* cycle time *of an RRG, τ(RRG), is the maximum delay of all combinational paths.*

Let us assume that combinational delays of nodes $F1$, $F2$, $F3$ are equal to one time unit while the delays of the rest of the nodes are equal to zero. Then, the cycle time of the RRG from the figure 1(a) is equal to three time units. The combinational path $F1, F2, F3, f, m$ is *critical*. Its delay is equal to the cycle time of the RRG.

Definition 2.4 (Throughput) *Given an RRG, the* throughput*, $\Theta(n)$, of node $n \in N$ is: $\Theta(n) = \lim_{t\to\infty} \frac{\sigma_n(t)}{t}$, where $\sigma_n(t)$ is the number of tokens produced by n till time stamp t.*

Given an RRG the throughput of every node is the same [10], i.e., $\Theta(n_i) = \Theta(n_j)$ for every $n_i, n_j \in N$. Thus, the throughput of any node can be denoted by Θ(RRG).

Notice that if an RRG has no bubbles (see Figure 1(a)), one token is produced by each EB each cycle, then $\Theta(\text{RRG}) = 1$.

Definition 2.5 (Effective cycle time) *Given an RRG, the* effective cycle time *is defined as*

$$\xi(\textit{RRG}) = \frac{\tau(\textit{RRG})}{\Theta(\textit{RRG})}.$$

Let us now define formally the RR transformations.

Definition 2.6 (Retiming vector) *Given an RRG, a retiming vector $r \in \mathbb{Z}^{|N|}$ is a map $N \to \mathbb{Z}$ that for every edge $e = (u, v)$ transforms R_0 to R'_0 as follows: $R'_0(e) = R_0(e) + r(v) - r(u)$.*

In contrast to the classical definition in [5] definition 2.6 allows negative values for R_0. This is because in ESs EBs can keep anti-tokens [3].

Definition 2.7 (RR configuration) *Given an RRG, a RR configuration,* RC*, is a pair of vectors $R'_0 \in \mathbb{Z}^{|E|}$, $R' \in \mathbb{Z}^{+|E|}$ that satisfies the following constraints:*

$$\begin{aligned} &R'_0(e) = R_0(e) + r(v) - r(u), \\ &R'(e) \geq R'_0(e), \textit{ for each edge } e = (u, v), \end{aligned} \tag{1}$$

where r is a retiming vector.

An RRG has a lot of different RCs. For instance, the retiming vector: $r(m) = -2, r(F1) = -2, r(F2) = -1, r(f) = r(F3) = 0$, transforms the RC in Figure 1(a) to the RC in Figure 2.

Combinational path constraints. In order for an RC to meet a cycle time τ, the delay of every combinational path in the corresponding RRG must be smaller than or equal to τ. The following lemma provides a simple way to verify this for a strongly connected RRG.

Lemma 2.1 (Cycle time constraints [9]) *Given a strongly connected RRG and a constant τ. The* RC *of the RRG has cycle time less than or equal to τ iff the following system of inequalities is feasible:*

$$\begin{aligned} & t^{in}(e) \geq t^{out}(e') + \beta(u) \qquad \forall e' = (w,u), e = (u,v) \\ & t^{out}(e) \geq t^{in}(e) - \tau^* \cdot R'(e) \\ & t^{out}(e) \geq 0, \quad t^{in}(e) \leq \tau. \end{aligned} \tag{2}$$

In the constraints, $t^{in}(e)$ and $t^{out}(e)$ are real variables. The constant τ^* must be large enough to ensure that its value is greater than any possible value of cycle time τ. A simple way to ensure this is to make τ^* equal to the sum of all combinational delays of the RRG. The extension of the lemma to non-strongly connected RRGs is trivial.

In the following, the constraints (2) for a given RC and cycle time τ will be represented by the predicate Path_Constraints(RC, τ).

3 Throughput of RRG

This section shows how the performance of an RRG can be estimated via LP. The estimation is based on results of [10] on performance analysis of systems with early evaluation. Let us recall some basic definitions from [10].

3.1 Guarded marked graphs

Guarded marked graphs were originally presented in [10]. They are very suitable for modeling of ESs with early evaluation nodes.

Let us introduce a *guard* concept into the classical definition of the *marked graph* [11].

Definition 3.1 (GMG) *A* Guarded Marked Graph *(GMG) is a tuple $\langle N, E, m_0, G\rangle$ where:*

- *N is the set of nodes which is partitioned into N_1 and N_2: N_1 includes the simple nodes and N_2 - the early evaluation nodes.*
- *$E \subset N \times N$ is the set of edges.*
- *$m_0 : E \rightarrow Z$ assigns an initial number of tokens (possibly negative), $m_0(e)$, to each edge e.*
- *$G : N \rightarrow 2^{2^E}$ assigns a set of guards to every node, such that the following condition is satisfied. Let us denote the set of input and output edges of a node n_i as ${}^\bullet n_i = \{(n_j, n_i) | (n_j, n_i) \in E\}$ and $n_i^\bullet = \{(n_i, n_j) | (n_i, n_j) \in E\}$, respectively. Then for $n \in N_1$ the guards set $G(n)$ is one element set $\{\{{}^\bullet n\}\}$. This means that all input edges of the node n are in the same guard. For $n \in N_2$ the guards set has $|{}^\bullet n|$ elements, $G(n) = \{{}^\bullet n\}$.*

Definition 3.2 (Firing semantics) *The dynamic behavior of an GMG is determined by the following firing rules:*

- Guard selection. *A guard $g(n) \in G(n)$ for the next firing of n is selected non-deterministically. The guard selection is trivial for simple nodes, since they only have one guard. For early evaluation nodes any guard in $G(n)$ can be selected.*
- Enabling. *If the guard $g(t)$ has been selected for the next firing of n, then the node n becomes enabled when every input edge $e \in g(t)$ has positive marking.*
- Firing. *An enabled node n at marking m can fire leading to another marking m' such that*

$$m'(e) = \begin{cases} m(e) - 1 & \text{if } e \in {}^\bullet n \setminus n^\bullet \\ m(e) + 1 & \text{if } e \in n^\bullet \setminus {}^\bullet n \\ m(e) & \text{otherwise} \end{cases}$$

3.2 Timed guarded marked graphs

In order to carry out performance analysis on GMGs a timing interpretation must be added to it. For this purpose a nonnegative real number $\delta(n)$ is associated with every node n of the GMG. Moreover, each guard is assigned the probability of being selected.

Definition 3.3 (TGMG) *A* Timed Guarded Marked Graph *(TGMG) is a tuple $\langle N, E, m_0, G, \delta, \gamma \rangle$ where:*

- *$\langle N, E, m_0, G \rangle$ is a GMG.*
- *$\delta : N \to \mathbb{R}^+$ assigns a nonnegative delay to every node.*
- *$\gamma : G \to \mathbb{R}^+ \setminus \{0\}$ assigns a strictly positive probability to each guard of $G(n)$. It must hold that that: $\sum_{e \in G(n)} \gamma(e) = 1$.*

For the time evolution of an TGMG it is assumed that the guard selection process has zero duration and that it respects the probabilities (γ) in any infinite execution. The *infinite servers* semantics is assumed [10].

Definition 3.4 (Steady state throughput) *The steady state throughput, $\Theta(\mathcal{N})$, of an TGMG is defined as:*

$$\Theta(\mathcal{N}) = \lim_{t \to \infty} \frac{\sigma(t)}{t} \tag{3}$$

where t represents the time and $\sigma(t)$ is the firing count vector at time t, i.e., the j's component of $\sigma(t)$ corresponds to the number of times node n_j has fired till the time stamp t.

Notice, $\Theta(\mathcal{N})$ is defined as a vector. In [10] it is shown that all transitions of an TGMG have the same throughput. The throughput is upper bounded by the solution of the following LP problem:

$$\begin{array}{l} Maximize \quad \phi : \\ \quad \delta(n) \cdot \phi \leq \widehat{m}(e), \qquad\qquad n \in N_1, e \in {}^\bullet n \\ \quad \delta(n) \cdot \phi \leq \sum_{e \in {}^\bullet n} \gamma(e) \cdot \widehat{m}(e), \quad n \in N_2 \\ \quad \widehat{m}(e) = m_0(e) + \sigma(u) - \sigma(v), \quad e = (u, v). \end{array} \tag{4}$$

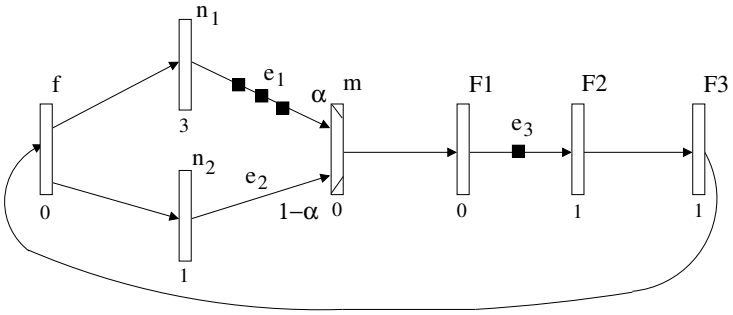

Figure 3: A timed guarded marked graph for the RRG 1(b).

The vector σ represents the firing count vector (with real components) that corresponds to the number of node firings from the initial marking, m_0, to the estimated average marking $\widehat{m}$.

Now it will be shown how to model RRGs with TGMGs to compute throughput bounds for RRGs by solving (4).

3.3 RRG throughput constraints

The next procedure shows how to construct a TGMG model for an RRG. (For illustration RRG from Figure 1(b) is used. Its TGMG is shown in Figure 3).

Procedure 1 (TGMG model for an RRG)
1. *Set the graph structure of the TGMG to be the same as the one of RRG.*
2. *A node of the TGMG is early evaluation iff its corresponding node in the RRG is. The branch selection probability function (γ) of the TGMG is equal to the corresponding function of the RRG.*
3. *If a node n of the RRG has one input edge e, then set: $m_0(e) = R_0(e), \delta(n) = R(e)$, e.g., the delay of the node F2 is equal to one and there is one token on the edge e_3.*
4. *If a node n has several input edges, then: a) add one node n_j on each input edge $e = (n_i, n)$ of the TGMG; b) set: $m_0(n_i, n_j) = 0, m_0(n_j, n) = R_0(e), \delta(n_j) = R(e)$; c) set $\delta(n) = 0$. For example, for the top (f, m) edge of the RRG the node n_1 with delay 3 is added to the TGMG; the delay of node m is equal to 0; there are three tokens on edge (n_1, m).*

If an RRG has only simple nodes its throughput is equal to the throughput of the TGMG obtained by Procedure 1. However, if an RRG has early evaluation nodes then the throughput of TGMG may be greater than the throughput of RRG. The following procedure refines a given TGMG model in order to alleviate this problem:

Procedure 2 (Refining a TGMG model)
1. *for every early evaluation node* $n \in N_2$ *do*
2. *add a new simple node* s*, set* $\delta(s) = 1$*,*
3. *add an edge* (n, s)*, set* $m_0(n, s) = 1$*,*
4. *for every input edge* $e = (n_i, n)$ *do*
5. *split* e *into two edges* (n_i, n_k) *and* (n_k, n)*,*
6. *add a new edge* (s, n_k)*,*
7. *set* $\delta(n_k) = 0$*,* $m_0(n_i, n_k) = m_0(e)$*,*
$m_0(n_k, n) = 0$*,* $m_0(s, n_k) = 0$*.*

Figure 4 shows a refined model for the TGMG from the figure 3. Basically, procedure 2 adds unit delay self loop for each early evaluation node n and then transforms TGMG to preserve the guard set $G(n)$ (see [10] for the details).

Lemma 3.1 *Given an RRG with throughput* Θ *and its corresponding TGMG that was obtained by the consecutive calls of procedures 1 and 2. Then the throughput of the TGMG is equal to the throughput of the RRG.*

From the lemma 3.1 and the inequalities (4) the following lemma follows:

Lemma 3.2 (Throughput constraints) *If the throughput of an RRG is equal to* Θ*, then the following system of inequalities is feasible:*

$$R(e) \leq x \cdot R_0(e) + \sigma(v) - \sigma(u),\ e = (u, v), v \in N_1, \tag{5}$$
$$R(e) \leq \sigma(u) - aux_R(e), \quad v \in N_2, \tag{6}$$
$$0 \leq \sum_{e \in {}^\bullet v} \gamma(e) \cdot (aux_0(e) - \sigma(v)), \tag{7}$$
$$x + \sigma(v) - \sigma(s) \geq 1, \tag{8}$$
$$\sigma(s) - aux_0(e) \geq 0, \tag{9}$$
$$x \cdot R_0(e) + aux_R(e) - aux_0(e) \geq 0. \tag{10}$$

where $x = 1/\Theta$ is used instead of Θ to avoid quadratic terms in (5). Inequalities (6)-(10) are applied for each early evaluation node v. Variables aux_R are the firing count for the nodes introduced by the Procedure 1. Variables aux_0 and s refer to the nodes introduced by Procedure 2. Let us denote the throughput constraints as $\texttt{Throughput\_Constraints}(\texttt{RC}, x)$ for a given RC and a constant $x \geq 1$. Using lemma 3.2 an upper bound of the throughput for a given RC, $\Theta^{lp}(\texttt{RC})$, can be computed with the following LP:

$$\begin{array}{l} Minimize \quad x : \\ \quad \texttt{Throughput\_Constraints}(\texttt{RC}, x). \end{array} \tag{11}$$

Let x_0 be a solution of (11) for a given RC, then $\Theta^{lp}(\texttt{RC}) = 1/x_0$. The corresponding effective cycle time will be denoted as $\xi^{lp}(\texttt{RC})$, i.e., $\xi^{lp} = \tau(\texttt{RC})/\Theta^{lp}(\texttt{RC})$, where $\tau(\texttt{RC})$ refers to the cycle time of the RC.

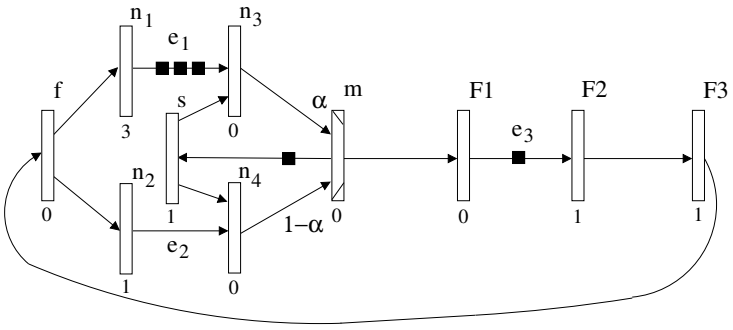

Figure 4: A refined TGMG model for the RRG 1(b).

4 Retiming and Recycling

A straightforward method that combines the combinational path (2) and throughput constraints (5)-(10) for minimizing the effective cycle time leads to the following non-convex mixed integer quadratic programming problem:

$$\begin{aligned} Minimize \quad & x \cdot \tau, \\ & R'_0(e) = R_0(e) + r(v) - r(u), \\ & R' \geq R_0, R' \geq 0, \\ & \texttt{Path\_Constraints}(\texttt{RC}, \tau), \\ & \texttt{Throughput\_Constraints}(\texttt{RC}, x), \\ & R' \in INT, r \in INT. \end{aligned} \tag{12}$$

Unfortunately, the exact solution of (12) is not necessarily the one with the minimum effective cycle time, $\texttt{RC}_{min}$, since (11) yields the throughput upper bound, not the exact value. On the other hand, the programming problem (12) represents a big challenge for existing solvers. Therefore, a heuristics based on a mixed integer linear programming (MILP) is next provided to solve (12). The heuristics makes use of the notion of *non-dominated* RC.

Definition 4.1 (Non-dominated RC) *Given an RRG and its two RCs* $\texttt{RC}_1$ *and* $\texttt{RC}_2$*, we say that* $\texttt{RC}_1$ *dominates* $\texttt{RC}_2$ *if* $\Theta(\texttt{RC}_1) > \Theta(\texttt{RC}_2)$ *and* $\tau(\texttt{RC}_1) \leq \tau(\texttt{RC}_2)$*. An* RC *is called* non-dominated *if there is no other* RC *that dominates it.*

Notice that $\texttt{RC}_{min}$ must be non-dominated.

The only non-linear part in (12) is the multiplication of the cycle time and the inversion of the throughput in the objective function. Fixing one of these variables leads to an MILP.

Let $\texttt{MIN\_CYC}(x)$ be the RC given by the solution of (12) with x being constant. In other words, $\texttt{MIN\_CYC}(x)$ is an RC with throughput greater or equal to $1/x$ and minimal cycle time. In particular, $\texttt{MIN\_CYC}(1)$ returns a min-delay retiming configuration. Similarly, let $\texttt{MAX\_THR}(\tau)$ denote the RC with cycle time less than or equal to τ and maximal throughput.

The following heuristics finds several RCs which are non-dominated with respect to the Θ^{lp} and returns one with the minimal effective cycle time, $\mathtt{RC}^{lp}_{min}$. It also returns k best RCs (in case $\mathtt{RC}^{lp}_{min}$ is not a minimal effective cycle configuration $\mathtt{RC}_{min}$).

The following observation helps to find such RCs. *Given an RRG and a positive constant* $\Theta \leq 1$*, let* $\tau_1 = \tau(\mathtt{MIN\_CYC}(1/\Theta))$ *be the cycle time of the* RC $\mathtt{MIN\_CYC}(1/\Theta)$. *Then, the* RC $\mathtt{MAX\_THR}(\tau_1)$ *is non-dominated.*

If an RRG has only simple nodes, the obtained RC is also non-dominated with respect to the actual throughput [9].

Let $\beta_{max} = \max_{n \in N} \beta(n)$ be the maximum combinational delay of an RRG. The following algorithm finds the $\mathtt{RC}^{lp}_{min}$:

```
MIN_EFF_CYC(RRG, k):
    τ = β_max, RC = RC^lp_min = MAX_THR(τ);
    store_nondominated_configuration(RC);
    while (Θ^lp(RC) < 1)
        Θ = Θ^lp(RC) + ε;
        τ = τ(MIN_CYC(1/Θ));
        RC = MAX_THR(τ);
        store_nondominated_configuration(RC);
        if (ξ^lp(RC) < ξ(RC^lp_min)) then RC^lp_min = RC;
    return RC^lp_min and k other best RC;
```

The *while* loop of the algorithm finds several non-dominated RCs and selects one with the minimal ξ^{lp}. The last stored RC is always a min-delay retiming configuration. The value of the constant ε must be at most the minimal difference between throughput of two RCs of the RRG. Notice that if one would find such constant, than MIN_EFF_CYC solves (12) exactly. Even in this case the $\xi(\mathtt{RC}^{lp}_{min})$, in general, is a lower bound for $\xi(\mathtt{RC}_{min})$. In the experiments the value of ε was set to 0.01.

5 Experimental results

A set of experiments was performed to verify the throughput model (5)-(10) and to demonstrate optimization power of the algorithm for ESs with early evaluation.

A set of RRGs has been generated as follows:

- The ISCAS89 circuits have been used[2] to extract the underlying graph structures. The largest strongly connected component of the each ISCAS89 circuit was taken. The rest of the nodes and edges were removed.
- Each edge was assigned a token with probability 0.25. Thus, about 25% of the edges were assigned an initialized register, whereas the rest were just wires.
- Each node was assigned a combinational delay uniformly distributed in the interval $(0, 20]$.

[2]The ISCAS89 benchmarks were used only for getting realistic graph structures.

- A node with more than one input was marked as early evaluated with probability of 0.4. The branch probabilities were selected randomly.

Table 1: All non-dominated RCs for the test case s526

Name	τ	Θ^{lp}	Θ	err(%)	ξ^{lp}	ξ	$\Delta(\%)$
s526	19.98	0.2500	0.2390	4.6025	79.9200	83.5983	
	24.10	0.3333	0.3050	9.2896	72.3000	79.0164	
	31.74	0.4936	0.4200	17.5219	**64.3041**	75.5714	
	56.54	0.8367	0.7910	5.7787	67.5742	**71.4791**	
	74.52	1.0000	1.0000	0.0000	74.5200	74.5200	5.4

For each test case the $\texttt{RC}^{lp}_{min}$ was found by using MIN_EFF_CYC. The Verilog representation of elastic controller was generated for each non-dominated RC. The actual throughput was calculated by performing intensive simulations.

Table 1 shows all the RCs that were found for the test case s526. Rows of the table correspond to different RCs. The column τ provides the cycle time of the RC. The columns Θ^{lp} and Θ provide the throughput upper bound and the actual throughput of the RC (obtained by simulation) respectively. The column $err(\%)$ provides the relative difference between the throughput upper bound Θ^{lp} and Θ. The effective cycle times of $\texttt{RC}^{lp}_{min}$ and $\texttt{RC}_{min}$ are marked in bold in the columns ξ^{lp} and ξ respectively. The last column $\Delta(\%)$ is the relative difference between $\xi(\texttt{RC}_{min})$ and $\xi(\texttt{RC}^{lp}_{min})$, e.g., for s526 it is equal to $(75.5714-71.4791)/71.4791\cdot 100\% \approx 5.4\%$. It can be seen that the $\texttt{RC}^{lp}_{min}$ and $\texttt{RC}_{min}$ are different configurations in this case, however $\texttt{RC}^{lp}_{min}$ has only 5.4% worse performance. Also the second best configuration returned by the algorithm does correspond to $\texttt{RC}_{min}$.

Table 2 shows the obtained results. The first column is the name of the underlying ISCAS89 circuit. The next three columns are the number of simple nodes, early evaluated nodes and edges respectively. The column ξ^* provides the cycle time of test case before the optimization (it is equal to the effective cycle time because originally RRGs have no bubbles). The column ξ_{nee} provides the minimal effective cycle time of the RRG with all nodes being simple (late evaluation). It often coincides with the min-delay retiming cycle time (see [9] for details). In the experiments the ξ_{nee} was always provided by min-delay retiming configuration. The columns ξ^{lp}_{min} and ξ^{sim}_{min} show $\xi(\texttt{RC}^{lp}_{min})$ and $\xi(\texttt{RC}_{min})$, respectively. E.g., for s526 the corresponding values are equal to 75.57 and 71.48. The last column $I(\%)$ provides the performance improvement obtained by MIN_EFF_CYC using early evaluation. It is calculated as follows: $I = ((\xi_{nee} - \xi^{sim}_{min})/\xi_{nee}) \cdot 100\%$.

CPLEX [12] was used as an MILP solver. The timeout for integer optimization was set to 20 minutes in all experiments.

Observation 1: The average effective cycle time improvement is equal to 14.5% (the average value of the column $I\%$). The improvement strongly depends from the position of early evaluation nodes. The ξ_{nee} was not improved for s832, s1488, s1494. This is because some critical directed cycles (the cycles where bubbles have to be inserted) have no early evaluation nodes. The early evaluation does not affect the performance os such ESs.

Observation 2: The $\texttt{RC}^{lp}_{min}$ coincides with $\texttt{RC}^{sim}_{min}$ in more than half of the exam-

Table 2: Experimental results.

Name	$\|N_1\|$	$\|N_2\|$	$\|E\|$	ξ^*	ξ_{nee}	ξ^{lp}_{min}	ξ^{sim}_{min}	$I\%$
s208	7	1	9	87.58	87.58	85.54	85.54	2.3
s641	206	15	270	183.15	109.62	93.72	89.98	17.9
s27	9	5	24	43.73	43.73	32.31	32.31	26.1
s444	45	13	82	174.88	106.75	92.50	92.50	13.3
s838	7	1	9	68.40	68.40	59.99	59.99	12.3
s386	36	12	131	74.80	74.60	58.55	59.81	21.5
s344	122	13	176	130.63	114.19	90.79	82.89	27.4
s400	37	9	66	149.29	79.50	80.10	77.63	2.3
s526	43	7	71	144.47	74.52	75.57	71.48	4.1
s382	35	7	60	84.65	68.47	66.07	66.07	3.5
s420	7	1	9	76.70	76.70	59.78	59.78	22.1
s832	76	41	462	62.11	50.39	50.39	50.39	0.0
s1488	85	48	572	64.28	59.52	59.52	59.52	0.0
s510	63	40	407	116.63	116.63	73.26	73.26	37.2
s953	232	36	371	354.86	292.28	125.92	119.53	59.1
s713	229	27	341	119.15	96.63	99.13	95.96	0.7
s1494	88	48	572	61.97	55.80	55.80	55.80	0.0
s820	72	38	424	55.64	53.23	46.90	46.90	13.5

ples. In s641,s386, s400,s526, s713, s953 the relative difference between $\xi(\mathrm{RC}_{min})$ and $\xi(\mathrm{RC}^{lp}_{min})$ ($\Delta(\%)$) is within 5%.

Observation 3: The average error $err(\%)$ of the thtoughput estimation is equal to 12.5%. It is calculated as the relative difference between Θ^{lp} and Θ. The error usually increases with the number of bubbles that were inserted in the RRG and achieves 35% for some configurations. Usually the error is proportional to the difference between throughputs of an RRG with and without early evaluation nodes.

6 Conclusions and future work

A quadratic mixed integer programming model for retiming and recycling of elastic systems with early evaluation nodes is presented. This model is solved using a heuristics based on Mixed Integer Linear Programming (MILP). In most cases the heuristics finds an exact solution. With the proposed approach the performace of a sequential circuit can be improved up to 50% with respect to the minimal cycle time achievable by the retiming technique.

The proposed MILPs are difficult to solve exactly for circuit graphs with more than one thousand edges. However, there are simple and efficient heuristics for solving MILP problems. Exploring such heuristics is a part of the future work.

The proposed model can be extended to handle *telescopic* nodes (i.e., nodes with variable combinational delays).

References

[1] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Transactions on Computer-Aided Design*, vol. 20, no. 9, pp. 1059–1076, Sep. 2001.

[2] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proc. ACM/IEEE Design Automation Conference*, Jul. 2006, pp. 657–662.

[3] J. Cortadella and M. Kishinevsky, "Synchronous elastic circuits with early evaluation and token counterflow," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 2007, pp. 416–419.

[4] T. Kam, M. Kishinevsky, J. Cortadella, and M. Galceran-Oms, "Correct-by-construction microarchitectural pipelining," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2008.

[5] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[6] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Coping with latency in SoC design," *IEEE Micro, Special Issue on Systems on Chip*, vol. 22, no. 5, p. 12, October 2002.

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

[8] R. W. Wolff, *Stochastic modeling and the theory of queues*. Prentice Hall, 1989.

[9] D. Bufistov, J. Cortadella, M. Kishinevsky, and S. Sapatnekar, "A general model for performance optimization of sequential systems," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2007.

[10] J. Julvez, J. Cortadella, and M. Kishinevsky, "Performance analysis of concurrent systems with early evaluation," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2006.

[11] F. Commoner, A. W. Holt, S. Even, and A. Pnueli, "Marked directed graphs," *Journal of Computer and System Sciences*, vol. 5, pp. 511–523, 1971.

[12] "CPLEX," Available from http://www.ilog.com.