Retiming and Recycling for Elastic Systems with Early Evaluation

Dmitry E. Bufistov
Universitat Politècnica de Catalunya
Barcelona, Spain

Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

Marc Galceran-OMs
Universitat Politècnica de Catalunya
Barcelona, Spain

Jorge Júlvez
University of 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.

Categories and Subject Descriptors: B.5.2 [Register-transfer-level implementation]: Design Aids.

General Terms: Design, Theory, Performance.

Keywords: Elastic systems, early evaluation, optimization.

1. INTRODUCTION

Latency-insensitive (a.k.a. synchronous elastic) systems tolerate changes in communication and computation latencies [5, 8]. The term “elastic system”, ES, will be used in this paper.

An ES 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 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.

Any synchronous circuit can be transformed into an equivalent ES following a simple automatic flow [6, 8].

A key aspect of ESs is that they accept a set of valid transforma-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and a fee.

DAC’09, July 26–31, 2009, San Francisco, California, USA
Copyright 2009 ACM 978-1-60558-497-3/09/07....10.00

1.1 Retiming and Recycling Graph

Retiming [11] is a well-known technique for sequential optimization. It moves registers across circuit 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 blocks labelled with EBs when needed. The RRG can be viewed as an extension of the retiming graph [11].

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 delays while other nodes have zero delays, 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.

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 directed cycle f, 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 [11]. Thus, the moves of retiming cannot reduce the cycle time in this example: 3 is the minimal cycle time achievable by retiming. Despite, the retiming cannot reduce the cycle time in this example: 3 is the number of EB bubbles reduce the throughput of the ES (defined as the amount of useful work done per cycle) to \( \frac{3}{4} \). The multiplexer 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 by using RR is the main goal of this paper.

### 1.3 Early evaluation (EE)

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 (EE)** 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. Recently, different schemes to handle arriving irrelevant tokens to avoid spurious enabling of functional units. The behavior of ESs with EE can be modeled using **Markov chains** [9]. Although this approach does not scale in general, 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 EE, 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 \( \frac{4}{3} \approx 0.833 \) that is about 16% better than the throughput for the ES from Figure 1(b) with an EE mux.

**The contributions.** The first contribution of this paper is the demonstration, as shown in the introductory example, that allowing anti-tokens in initial configurations may help to achieve a better throughput. This is not the case for ESs without EE.

The second contribution is a precise marked graph model for performance estimation of ESs with EE.

The last contribution is a method for minimization of the effective cycle time of ESs with EE. The work is an extension of the paper about performance optimization of ESs with late evaluation [3].

### 2. BACKGROUND

This section formalizes basic concepts.

A Retiming and Recycling Graph (RRG) is a tuple \( \langle S, \beta, R, R, \gamma \rangle \), 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 EE nodes. \( \beta : N \to \mathbb{R}^+ \) assigns combinational delay to each node. \( R_0 : E \to \mathbb{Z}^+ \) is the number of tokens on each edge. If negative, \( R_0 \) is the number of anti-tokens. To ensure liveliness the sum of tokens on each directed cycle of \( S \) must be positive. \( R : E \to \mathbb{Z}^+ \) is the number of EBs on each edge, condition \( R \geq R_0 \) must hold. \( \gamma : E \to \mathbb{R} \) assigns branch selection probability for input edges of EE nodes \( n \in N_2 \). The sum of the probabilities for all inputs of an EE node \( n \in N_2 \) is equal to one. As an example, the values of \( R_0, R \) and \( \gamma \) of the bottom (edge) \((f, m)\) of the RRG in Figure 1(b) are \( 3, 3 \) and \( \alpha \) \((0, 1 \text{ and } 1 - \alpha)\).

Given an RRG, a **combinational path** is a sequence of adjacent edges \( e_1, \ldots, e_k \) such that \( R(e_i) = 0, 1 \leq i \leq k \). The delay of the combinational path is the sum of the delays of the corresponding nodes. For example, the path formed by the nodes \( F_1, F_2, F_3 \) in Figure 1(a) is combinational while the path \( f, m, F_1 \) is not.

The **cycle time** of an RRG, \( \tau(RRG) \), is the maximum delay of all combinational paths.

Let us assume that combinational delays of nodes \( F_1, F_2, F_3 \) are equal to 1 time unit while the delays of the rest of the nodes are equal to 0. Then, the cycle time of the RRG in Figure 1(a) is equal to 3. The combinational path \( F_1, F_2, F_3, f, m \) is **critical**. Its delay is equal to the cycle time of the RRG.

The **throughput**, \( \Theta(n_\alpha) \), of node \( n \in N \) of an RRG is defined as: \( \Theta(n_\alpha) = \lim_{t \to \infty} \frac{\sum_{n(t)} \gamma_n(t)}{t} \), where \( \gamma_n(t) \) is the number of tokens produced by \( n \) till time stamp \( t \). The throughput of every node is the same [9], i.e., \( \Theta(n_\alpha) = \Theta(n_j) \) for every \( n_i, n_j \in N \). Thus, the throughput of any node can be denoted by \( \Theta(RRG) \).

Notice that if an RRG has no bubbles (see Figure 1(a)), one token is produced by each EB each cycle, then \( \Theta(RRG) = 1 \). The **effective cycle time of a RRG**, \( \theta(RRG) \), is the ratio of the cycle time and the throughput.

A retiming vector \( \tau \in \mathbb{Z}^{|N|} \) of a given RRG, is a map \( N \to \mathbb{Z} \) that for every edge \( e = (u, v) \) transforms \( R_0 \) to \( R_0' \) as follows: \( R_0'(e) = R_0(e) + \tau(v) - \tau(u) \).

In contrast to the classical definition in [11] this definition allows negative values for \( R_0 \). This is because in ESs EBs can keep anti-tokens [7].

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:

\[
R_0'(e) = R_0(e) + \tau(v) - \tau(u), \\
R'(e) \geq R_0'(e), \text{ for each edge } e = (u, v),
\]

(1)
where $\tau$ 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 R in Figure 1(a) to the RC in Figure 2.

**Combinational path constraints.** In order for an RC to meet a cycle time $\tau$, the delay of every combinational path in the corresponding RRG must be smaller than or equal to $\tau$. There are a set of linear constraints that guarantees this [3].

In the following, these constraints for a given RC and cycle time $\tau$ will be referred as Path Constr(RC, $\tau$).

### 3. THROUGHPUT OF RRG

The performance of an RRG can be estimated by using the result from [9] on performance analysis of guarded marked graphs.

A Guarded Marked Graph (GMG) is a tuple $(N, E, m_0, G)$ where $N$ is the set of nodes which is partitioned into subsets $N_1$ and $N_2$: $N_1$ includes the simple nodes and $N_2$ - the EE nodes; $E \subseteq N \times N$ is the set of edges; $m_0 : E \rightarrow Z$ assigns an initial number of tokens (possibly negative), $m_0(e)$, to each edge $e$; $G : N \rightarrow 2^E$ 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 $n_i^* = \{(n_j, n_i) \mid (n_i, n_j) \in E\}$ and $n_i = \{(n_i, n_j) \mid (n_i, n_j) \in E\}$, respectively. Then for $n \in N_1$ the guards set $G(n)$ is one element set of $\{n\}$. This means that all input edges of the node $n$ are in the same guard. For $n \in N_2$ the guards set has $n$ elements, $G(n) = \{n\}$.

The behavior of an GMG is determined by the following rules:

1. **Guard selection.** A guard $g(n) \in G(n)$ for the next firing of $n$ is selected nondeterministically. The guard selection is trivial for simple nodes, since they only have one guard. For EE nodes any guard in $G(n)$ can be selected.

2. **Enabling.** If the guard $g(n)$ has been selected for the next firing of $n$, then the node $n$ becomes enabled when corresponding input edge has positive marking.

3. **Firing.** An enabled node $n$ at marking $m$ can fire leading to another marking $m'$ by removing one token from each input edge and adding one token to each output edge.

**Timed guarded marked graphs.** In order to carry out performance analysis on GMGs a timing interpretation must be added to it. Each guard must be assigned a probability of being selected.

A Timed Guarded Marked Graph (TGMG) is a tuple $(N, E, m_0, G, \delta, \gamma)$ where $(N, E, m_0, G)$ is a GMG; $\delta : N \rightarrow \mathbb{R}^+$ assigns a nonnegative delay to every node; $\gamma : G \rightarrow \mathbb{R}^+ \setminus \{0\}$ assigns a strictly positive probability to each guard of $G(n)$. It must hold 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 ($\gamma$) in an infinite execution.

The throughput, $\Theta(N)$, of an TGMG is defined as: $\Theta(N) = \lim_{t \to \infty} \frac{\sigma_t}{t}$, 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 $t$.

Notice, $\Theta(N)$ is defined as a vector. In [9] it is shown that all nodes of an TGMG have the same throughput. The throughput is upper bounded by the solution of the following LP problem:

$$\begin{align*}
\text{Maximize} & \quad \phi : \\
\delta(n) \cdot \phi & \leq \bar{m}(e), \quad n \in N_1, e \in n^* \\
\delta(n) \cdot \phi & \leq \sum_{e \in n^*} \gamma(e) \cdot \bar{m}(e), \quad n \in N_2 \\
\bar{m}(e) & = m_0(e) + \sigma(u) - \sigma(v), \quad e = (u, v).
\end{align*}$$

The vector $\sigma$ is real in the constraints.

**RRG throughput constraints.** There is a simple procedure that constructs a TGMG model for an RRG. Because of the lack of the space the formal description is skipped. It can be found in [4]. For illustration the initial TGMG model for the RRG in Figure 1(b) is shown in Figure 3(a). Figure 3(b) shows the final version. Basically, a unit delay self loop for each EE node has been added and then TGMG was transformed to preserve the guards set $G(n)$ [4].

Applying (2) to the TGMG model of an RRG it can be guaranteed with the linear constraints that a given RC has throughput upper bound $1/x, x \geq 1$ [4]. Let us denote the set of this constraints as Thr Constr(RC, $x$). The throughput upper bound of a given RC can be found as a minimum value of $x$ subject to the Thr Constr(RC, $x$). Let us denote this upper bound as $\Theta^U(RC)$ and the corresponding effective cycle time as $\xi^U(RC)$, i.e., $\xi^U = \tau(RC)/\Theta^U(RC)$, where $\tau(RC)$ refers to the cycle time of the RC.

### 4. RETIMING AND RECYCLING

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

$$\begin{align*}
\text{Minimize} & \quad x \cdot \tau, \\
R_0(e) & = R_0 + r + r(u) - r(u), \\
R_0' & \geq R_0, R_0' \geq 0, \\
\text{Path Constr}(RC, \tau), \\
\text{Thr Constr}(RC, x), \\
R' & \in INT, r \in INT.
\end{align*}$$

The exact solution of (3) is not necessarily the one with the minimum effective cycle time, $RC_{min}$, but it is a very good approximation. On the other hand, (3) represents a big challenge for existing solvers. [4] provides a heuristics based on a MILP to solve (3). This heuristics finds few non-dominated RCs and select one with the minimal effective cycle time, $RC^p_{min}$.

### 5. EXPERIMENTAL RESULTS

A set of experiments was performed to verify the throughput model and to demonstrate optimization power of the algorithm for ESs with EE.

A set of random RRGs has been generated. The ISCAS89 circuits have been used to extract the underlying graph structures. All

---

1Backward arcs modeling backpressure can be eliminated by buffer sizing [12]
parameters have been set randomly, based on simple criterions. For each test case the RC with the minimal effective cycle time, \( \text{RC}_{\text{min}} \), was found. 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 non-dominated RCs for the test case s526. Rows of the table correspond to different RCs. The column \( \tau \) provides the cycle time of the RC. The columns \( \Theta(l^p \text{ and } \Theta) \) provide the throughput upper bound and the actual throughput of the RC (obtained by simulation) respectively. The column \( \text{err}(\%) \) provides the relative difference between the throughput upper bound \( l^p \text{ and } \Theta \) and the effective cycle times of \( \text{RC}_{\text{min}} \) and \( \text{RC}_{\text{min}} \) are marked in bold in the columns \( l^p \text{ and } \Theta \) respectively. The last column \( \Delta(\%) \) is the relative difference between \( \xi(\text{RC}_{\text{min}}) \) and \( \xi(\text{RC}_{\text{min}}) \), e.g., for s26 it is equal to \((75.5714 - 71.4791)/71.4791 \times 100 \approx 5.4\% \). It can be seen that the \( \text{RC}_{\text{min}} \text{ and } \text{RC}_{\text{min}} \) are different configurations in this case, however \( \text{RC}_{\text{min}} \) has only 5.4\% worse performance. Also the second best configuration returned by the algorithm does correspond to \( \text{RC}_{\text{min}} \).

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, EE nodes and edges respectively. The column \( \xi \) provides the cycle time before the optimization (it is equal to the effective cycle time because originally RRGs have no bubbles). The column \( \xi_{\text{nee}} \) 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 [3] for details). In the experiments the \( \xi_{\text{nee}} \) was always provided by mindelay retiming configuration. The columns \( \xi_{\text{min}} \text{ and } \xi_{\text{min}} \) show \( \xi(\text{RC}_{\text{min}}) \text{ and } \xi(\text{RC}_{\text{min}}) \), respectively. E.g., for s26 the corresponding values are equal to 75.75 and 71.48. The last column \( I(\%) \) provides the performance improvement obtained by the proposed method with respect to EE without EE. It is calculated as follows: \( I = (\xi_{\text{nee}} - \xi_{\text{min}})/\xi_{\text{nee}} \times 100\% \). CPLEX was used as an MILP solver. The timeout for integer optimization was set to 20 minutes in all experiments. For all MILPs the optimal solutions were always found.

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 EE nodes. The \( \xi_{\text{nee}} \) was not improved for s382, s1488, s1494. This is because some critical directed cycles (the cycles where bubbles have to be inserted) have no early evaluation nodes. The EE does not affect the performance of such ESs.

Observation 2: The \( \text{RC}_{\text{min}} \) coincides with \( \text{RC}_{\text{min}} \) in more than half of the examples. In s641, s386, s400, s526, s713, s953 the value of \( \Delta(\%) \) is within 5\%.

Observation 3: The average error \( \text{err}(\%) \) of the throughput estimation is equal to 12.5\%. The error achieves 35\% for some configurations. Usually the error is proportional to the difference between throughputs of an RRG with and without EE nodes.

6. CONCLUSIONS AND FUTURE WORK
A MILP based algorithm for retiming and recycling of elastic systems with early evaluation has been presented. 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).

Acknowledgements. This research has been supported by FPU grant AP2005-4866, FI grant B1 00063, Juan de la Cierva fellowship from the Spanish Ministry of Education and Science, a grant from Intel Corp. CYCIT TIN 2004-07925 and research project CYCIT TIN2007-66523.

7. REFERENCES