Synchronous Elastic Circuits with Early Evaluation and Token Counterflow

Jordi Cortadella*
Universitat Politecnica de Catalunya
Barcelona, Spain

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

ABSTRACT

A protocol for latency-insensitive design with early evaluation is presented. The protocol is based on a symmetric view of the system in which tokens carrying information move in the forward direction and anti-tokens canceling information move in the backward direction. An implementation of the protocol and an example illustrate the flow for converting a regular synchronous design into an elastic circuit with early evaluation.

Categories and Subject Descriptors: B.5.2 [Register-transfer-level implementation]: Design Aids.
General Terms: Design, Theory, Verification.
Keywords: Elastic designs, protocols, synthesis.

1. INTRODUCTION

Synchronous elastic (or latency insensitive) systems have been suggested by a few research groups as a form of discretized asynchronous systems (see, e.g., [3, 5, 6]). Such systems are "elastic" in the sense that they can tolerate changes in the latencies of their components.

Conventional synchronous elastic systems are based on strict evaluation: the computation is initiated only when all input data are available. However this requirement is not always necessary. As an example, let us consider a multiplexer with the following behavior: z = `if` s `then` a `else` b. Early evaluation can be applied if, for instance, s and a are available and the value of s is *true*. In that case, the result $z = a$ can be produced and the value of b can be discarded when it arrives at the multiplexer.

In early evaluation, care must be taken in preventing the spurious enabling of functional units when the *non-required* inputs arrive later than the completion of the computation. A possible technique is the use of *negative tokens*, also called *anti-tokens*. Each time an early evaluation occurs, an anti-token is generated at every non-required input in such a way that when it meets the positive token they annihilate.

The idea of anti-tokens was used in [8,11], extending Petri nets for handling OR causality and nodes with arbitrary guard functions. [7] used a similar technique for performance estimation of systems with early evaluation. Early evaluation [10] and anti-tokens [1, 2] have also been used in asynchronous circuits.

This paper presents a behavioral model for synchronous elastic systems with early evaluation, called a *dual marked graph*. An implementation with a symmetry between the logic to handle tokens and anti-tokens is proposed. The paper concludes with an example illustrating the elasticization flow, the performance analysis of early evaluation, and system performance vs. controller area trade-offs.

2. MODEL FOR EARLY EVALUATION

This section presents a concurrent model for systems with early evaluation. We assume the reader to be familiar with the basic Petri net theory and refer to [9] for an excellent survey. The notation used in this paper is next presented.

A *marked graph* (MG) is a triple $G = (N, A, M_0)$, where N is a set of nodes, A is a set of arcs and $M_0 : A \to \mathbb{N}$ is a marking that assigns an initial number of tokens to each arc. Given a node n, the notation ${}^\bullet n$ and $n^\bullet$ is used to denote the set of incoming and outgoing arcs of n, respectively. Given a subset $\phi \subseteq A$, the total number of tokens of the arcs in ϕ at a given marking M is denoted by $M(\phi)$.

Without loss of generality, we model elastic systems with strongly connected MGs (SCMG). 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. We next review a few properties of SCMGs [9]:

Token preservation. Let ϕ be a cycle of an SCMG. For every reachable marking M, $M(\phi) = M_0(\phi)$.

Liveness. An SCMG is *live* if every cycle, ϕ, is marked positively at M_0, i.e., $M(\phi) > 0$.

Repetitive behavior. A firing sequence σ from a marking M leads to the same marking *iff* every node from N fires the same number of times in σ.

2.1 Dual marked graphs

We extend the class of MGs by allowing negative markings and early enabling. We call this class *dual marked graphs* (DMG). In DMGs, a marking M is a mapping $M : A \to \mathbb{Z}$. A subset of nodes $E \in N$ is declared to be *early-enabling*

*This work has been partially supported by a grant from Intel Corp., CICYT TIN2004-07925 and a Distinction by the Generalitat de Catalunya.

DAC 2007, June 4–8, 2007, San Diego, California, USA.

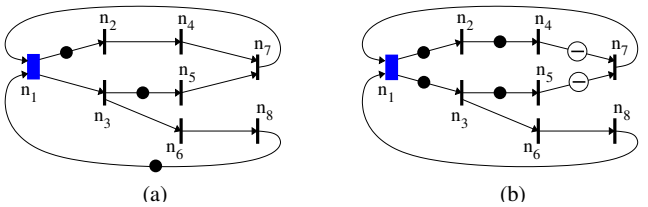

Figure 1: Dual marked graph: (a) initial marking, (b) reachable marking with anti-tokens.

(denoted with thicker bars). Given a marking M and a node n, the *enabling rules* for DMGs are defined as follows:
Positive (P) enabling: $M(a) > 0$ for every $a \in {}^\bullet n$. This is the conventional enabling condition.
Negative (N) enabling: $M(a) < 0$ for every $a \in n^\bullet$, i.e. all the successor arcs have negative tokens.
Early (E) enabling, for $n \in E$: $M({}^\bullet n) > 0$ and $M(a) \leq 0$ for some $a \in {}^\bullet n$, i.e. only some predecessor arcs have tokens.

E-enabling is only defined for early-enabling nodes. It models computations that can start without having all the incoming data available. E-enabling is usually associated with an external guard that depends on data values, e.g. a select signal of a multiplexer or a zero flag for an input operand of a multiplier.

Regardless of the enabling condition, the *firing rule* for DMGs is the same rule used for MGs. When an N-enabled node fires, it *propagates* the anti-tokens from the successor to the predecessor arcs. We call this phenomenon *token counterflow*. When an E-enabled node fires, it *generates* anti-tokens in the predecessor arcs that had no tokens.
Example. Fig. 1 depicts a DMG with one early-enabling node n_1 and three simple cycles: $C_1 = \{n_1, n_2, n_4, n_7\}$, $C_2 = \{n_1, n_3, n_5, n_7\}$ and $C_3 = \{n_1, n_3, n_6, n_8\}$. Every cycle has a token in the initial marking. Figure 1(b) depicts a reachable marking, with the symbol $\ominus$ representing anti-tokens. This marking can be reached from the initial one in Fig. 1(a) by firing nodes n_2 (P-enabling), n_1 (E-enabling) and n_7 (N-enabling). The firing preserves the sum of tokens at each cycle, considering positive and negative tokens.
Algebraic properties. Given that DMGs have the same firing rule as MGs, some fundamental properties also hold: *token preservation*, *liveness* and *repetitive behavior*.

All these properties are essential to guarantee a correct behavior in elastic systems with early evaluation.

3. ELASTICITY WITH ANTI-TOKENS

The implementation of elasticity with anti-tokens is based on the formal model of DMGs presented in Sect. 2 and the implementation of the SELF elastic protocol presented in [5]. More details can be found in [4].

In DMGs there are two flows, one for tokens and another for anti-tokens. A DMG can be split into two dual MGs, one for each flow. When a token and an anti-token are held in dual arcs, they must cancel each other.

The scheme of a linear pipeline is depicted in Fig. 2. The symbols V^+, S^+, V^- and S^- denote the *valid* and *stop* signals for the positive and negative flows. Note that the V^- signal has the semantics of a *Kill* for the positive tokens. The controllers are built based on the ordinary EHB depicted in Fig. 7(b) of [5] (also Fig. 3 of [4]). The negative component (bottom) is symmetric to the positive component (top). The dark gates are the only addendum to the

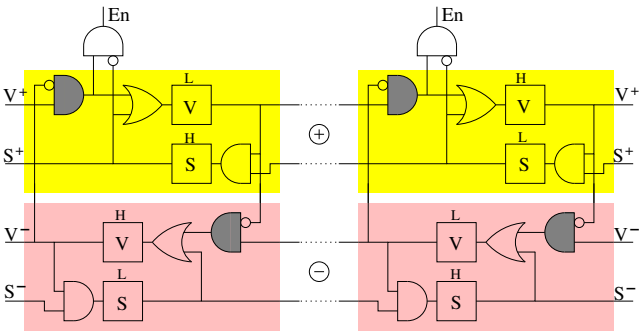

Figure 2: Linear pipeline with dual EHBs.

controllers and they play the role of the mutual cancellation of tokens and anti-tokens when they meet.

The abutment of elastic controllers with dual flow can easily lead to netlists with combinational cycles if controllers are not properly designed. For this reason, the gates cancelling tokens are placed at the boundaries of the EHB controllers, before the V and S signals are stored in their corresponding latches, as shown in Fig. 2.

The protocol for elastic communication implies certain invariants in the state of a channel. In particular:

$$\overline{V^- \wedge S^+} \qquad \text{and} \qquad \overline{V^+ \wedge S^-} \tag{1}$$

The first invariant indicates that it is not possible to kill a token and to stop it at the same time. The second invariant has a dual semantics for anti-tokens.

3.1 Join and fork controllers

The most difficult part of the implementation of elasticity is the correct synchronization of multiple flows by the *join* and *fork* structures (see Fig. 3(a) and 3(b)). The shadowed boxes of the join controller correspond to the basic join and fork controllers shown in Fig. 8 of [5] (also Fig. 4 of [4]). A join of tokens has a dual fork for anti-tokens, and vice-versa. The gates with label `I` are included to preserve the invariants specified by the expressions in (1). Finally, the gate `B` prevents a new transfer to occur before all the pending anti-tokens stored in the flip-flops FF have been propagated. Note that Figs. 3(a) and 3(b) are perfectly symmetric.

3.2 Join controller with early evaluation

The most important controller in this work is the one that generates anti-tokens. It is the *join with early evaluation*, depicted in Fig. 3(c). The design is similar to the join controller in Fig.3(a), with two important differences:

- The gates with label `G` generate the anti-tokens. For every channel, they implement the equation $\overline{V_{in}^+} \wedge V_{out}^+ \wedge \overline{S_{out}^+}$ that feeds the OR gate producing the anti-tokens at V_{in}^-. This equation is asserted when there is a transfer at the output channel without data at the input channel.

- The shadowed block with label `EE` implements the early evaluation function using the V_{in}^+ signals and some data coming from the data-path. This function substitutes the conjunction of V_{in}^+ in the conventional join and may be asserted even though not all V_{in}^+ signals are true.

Example. A join controller for a 2-input multiplexer would have three input channels: s, a, and b. The first channel would be associated with the select signal. Since every

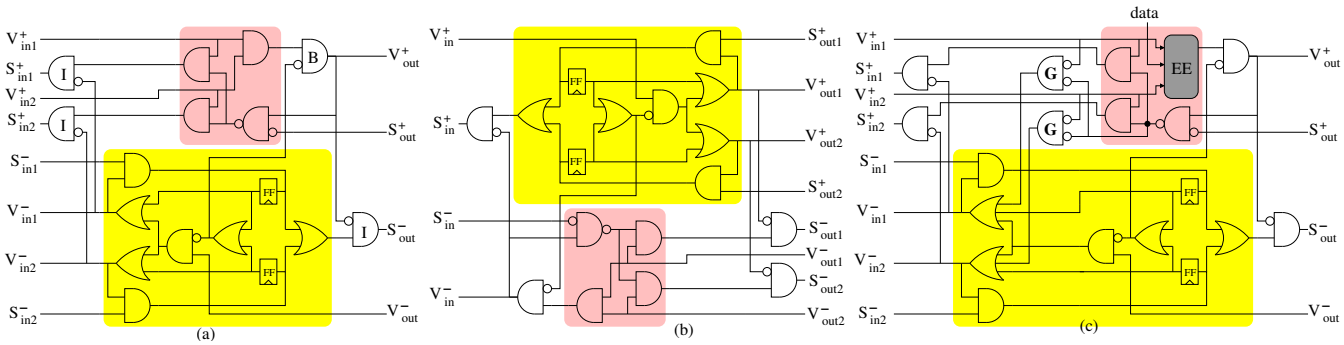

Figure 3: Dual elastic controllers: (a) Join, (b) Fork and (c) Join with early evaluation.

channel is elastic, it would also carry the V and S signals of the elastic protocol. The enabling function (block EE) could be $\texttt{EE} = V_s^+ \wedge ((s \wedge V_a^+) \vee (\overline{s} \wedge V_b^+))$. The signal s corresponds to the data value of the channel with the same name. Note that V_s^+ must always be true for the enabling of the module. An early enabling is produced, for example when $V_s^+ = V_a^+ = 1$ and $s = 1$. In this case, if $V_b^+ = 0$, an anti-token will be produced in the channel b.

3.3 Advanced features and verification

The elasticity with anti-tokens can be enhanced with some advanced features. In particular, the propagation of anti-tokens can be stopped (passive anti-tokens), thus saving area and energy to handle early evaluation. The controllers can also be modified to handle variable-latency units.

All the controllers presented in this paper have been verified using model checking. More details about these features and the verification strategy can be found in [4].

4. EXAMPLE

We will use the example in Fig. 4(a) to illustrate early evaluation. The system has five functional units: S, I, F, M and W. Registers are represented by shadowed boxes. Units F and W have three pipe-stages (not explicitly shown in the figure), while units S and I are not pipelined. Unit M is composed of two variable-latency multi-cycle units M_1 and M_2, delivering the result into a register. Unit S sends data to units I, F and M in parallel. Additionally, it sends control data (e.g. the opcode) to the register C. Unit W is a multiplexer that selects only one of the results according to the opcode. The selection probabilities are $0.6, 0.3$ and 0.1 for I, F and M, respectively.

The elastic conversion of this example (an automated process) proceeds in the following steps: (1) All registers in the data-path are transformed into pairs of master-slave latches with independent enable signals, and (2) the elastic control layer (see Fig. 4(b)) that generates the enable signals for the latches of the datapath is built using the components presented in [5] and Sect. 3 as follows:

- For every register in the datapath, an EB controller (composition of two EHBs) is included in the control layer.
- For every block, a controller with a *join* (J) or *early join* (EJ) at the inputs and a *fork* (F) at the outputs is built. The join or fork components are omitted if the block has only one input or one output, respectively. In the example we chose to use an early join for module W to exploit the early evaluation of W's multiplexer. The operations are encoded with two control signals, s_1 and s_2, as follows: 00 for I, 01 for F and 1- for M, thus resulting in the following early-enabling function for W:

  $$\texttt{EE} = V_c^+ \wedge ((\overline{s_1} \wedge \overline{s_2} \wedge V_I^+) \vee (\overline{s_1} \wedge s_2 \wedge V_F^+) \vee (s_1 \wedge V_M^+)).$$

  The control signals s_1 and s_2 are bundled with the valid signal V_c^+ from register C.
- A variable-latency controller (VL) is included for each variable-latency unit (M_1 and M_2). Both of them have a three-wire (go, done, ack) interface with the data-path.
- The connection of the controllers with the $\{V^+, S^+, V^-, S^-\}$ interfaces is done according to the connectivity of the corresponding units in the datapath. Solid arcs represent pairs of $\{V^+, S^+\}$ wires in the positive sub-channels, while dotted arcs going in the opposite direction represent pairs of $\{V^-, S^-\}$ wires in the negative sub-channels. Some of the channels (e.g. $W \rightarrow S$), do not have a negative part, since both V^- and S^- signals are constant 0. This simplification is performed by simple logic synthesis techniques.
- In this example we assume that the environment has only $\{V^+, S^+\}$ interface and does not attempt to kill any tokens inside the system.

4.1 Synthesis and simulation

A complete framework for elastic systems has been designed. It can generate models for simulation, verification and logic synthesis.

The Verilog simulation model incorporates statements to randomly generate the values of the control signals according to the probability distributions defined by the user. Similarly, it also generates random delays for the variable-latency units. In the example, we define a latency for M_1 of 2 and 10 cycles with probabilities 0.8 and 0.2, respectively. The delay for M_2 is defined as 1 or 2 cycles with probability 0.5 each. Initially all three EBs at the output of W have valid data, represented by the tokens at the registers. The other EBs have bubbles (invalid data).

Table 1 summarizes the results of 10K-cycle simulations and logic synthesis using SIS for five different configurations. The second column shows the throughput of the system measured as the number of transfers per cycle at the interfaces with the environment.

The throughput of a channel is computed as the sum of *positive transfers* ($V^+ \wedge \overline{S^+} \wedge \overline{V^-}$), *negative transfers* ($V^- \wedge \overline{S^-} \wedge \overline{V^+}$) and *kill cycles* ($V^+ \wedge V^-$) and *is the same for all system channels*. This is a direct consequence of the *repetitive behavior* of SCDMGs. The next five columns show the throughput of positive transfers (+), negative transfers

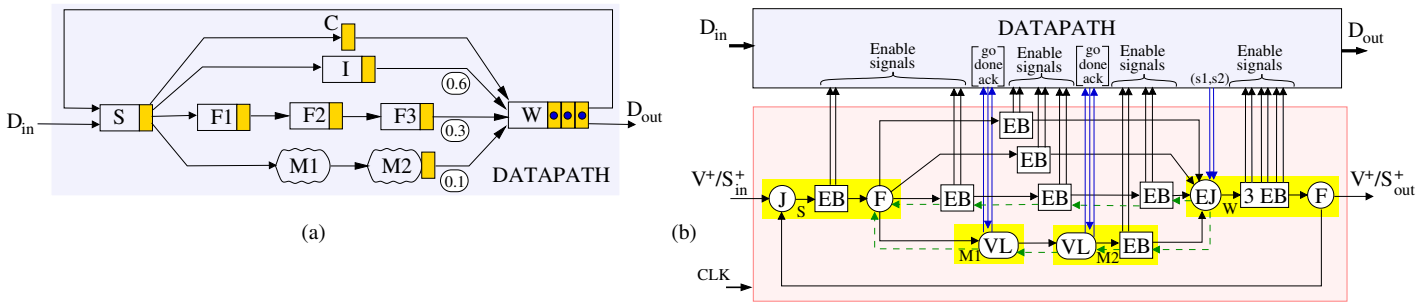

Figure 4: Example: (a) datapath, (b) elastic control.

Configuration	Th	$Th(F_2 \to F_3)$ +	$Th(F_2 \to F_3)$ ±	$Th(F_3 \to W)$ +	$Th(F_3 \to W)$ −	$Th(S \to M_1)$ +	$Th(S \to M_1)$ ±	$Th(M_1 \to M_2)$ +	$Th(M_1 \to M_2)$ −	$Th(M_2 \to W)$ +	$Th(M_2 \to W)$ −	Area lit	Area lat	Area ff
Active anti-tokens	0.400	0.205	0.195	0.132	0.268	0.328	0.071	0.328	0.071	0.204	0.195	253	56	9
No buffer ($S \to W$)	0.343	0.116	0.227	0.106	0.237	0.285	0.058	0.285	0.058	0.228	0.115	241	52	9
Passive ($F_3 \to W$)	0.387	0.387	0.000	0.387	0.000	0.318	0.069	0.318	0.069	0.280	0.107	213	44	9
Passive ($M_2 \to W$)	0.280	0.143	0.137	0.095	0.185	0.280	0.000	0.280	0.000	0.280	0.000	234	52	9
No early evaluation	0.277	0.277	0.000	0.277	0.000	0.277	0.000	0.277	0.000	0.277	0.000	176	40	6

Table 1: Throughput for different configurations of the example in Fig. 4.

(−) and kills (±) for five selected channels. The absent columns (e.g. ± in channel $F_3 \to W$) indicate that all values are 0. The last column shows the number of literals, transparent latches and flip-flops after logic synthesis.

The first configuration (active anti-tokens) corresponds to the one in Fig. 4(b): channel $F_3 \to W$ transfers anti-tokens 26.8% of the cycles, whereas channel $F_2 \to F_3$ kills tokens 19.5% of the cycles and has no anti-token transfers. The difference 26.8% − 19.5% = 7.3% is the fraction of tokens that are killed on the internal channel between the two EHBs at the output of F_3.

The second line corresponds to a configuration that has no buffer on the channel $S \to W$. Interestingly, this degrades the throughput from 0.4 to 0.343 since long operations in the pipeline prevent S from producing new values for channel $S \to W$. This phenomenon occurs when there is a large mismatch among the latencies of different branches in a pipeline. The buffer C mitigates this phenomenon.

The third and fourth lines report results with passive anti-tokens on one of the channels. This reduces the complexity of the control since some of the logic can be eliminated at the cost of some degradation in performance.

The numbers for $S \to M_1$ and $M_1 \to M_2$ are exactly the same, except for the fact that all negative tokens are transferred in the latter and killed in the former. This is because tokens are only killed at the boundaries with latches (see Fig. 2) and there are no latches between M_1 and M_2.

The last line reports a base-line performance for the lazy version of the control, in which the early join (EJ) is replaced by a regular join (J). In this case, no anti-tokens are held.

This experiment illustrates the potential impact of early evaluation in throughput and control complexity. It also shows that the area overhead of the control layer is small for wide (e.g. 32 or 64-bits) datapaths.

5. CONCLUSIONS

This paper describes a particular implementation of early evaluation, but many variations are possible. For example, it would be possible to extend the approach to store multiple anti-tokens at every controller. This might improve performance in some cases, but we found little experimental motivation for this feature. The propagation of anti-tokens creates a distributed memory within the system and eliminates the need for storing them within early join nodes.

The mechanism for anti-token counter-flow can also be used for handling exceptions inside elastic pipelines. For example, flushing a pipeline on branch mispredictions can be done by injecting anti-tokens.

This paper has focused on the performance aspects of the early evaluation. The power reduction aspect due to disabling non-required activity in the data-path is an equally important side effect of the anti-token counterflow.

6. REFERENCES

[1] M. Ampalam and M. Singh. Counterflow pipelining: Architectural support for preemption in asynchronous systems using anti-tokens. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 611–618, 2006.

[2] C. Brej. *Early Output Logic and Anti-Tokens*. PhD thesis, University of Manchester, 2005.

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

[4] J. Cortadella and M. Kishinevsky. Synchronous elastic circuits with early evaluation and token counterflow. Technical Report LSI-07-13-R, 2007. www.lsi.upc.edu/~techreps/files/R07-13.zip.

[5] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proc. Design Automation Conf.*, pages 657–662, July 2006.

[6] A. Edman and C. Svensson. Timing closure through a globally synchronous, timing partitioned design methodology. In *Proc. Design Automation Conference*, pages 71–74, 2004.

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

[8] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-Timed Design*. John Wiley & Sons, 1994.

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

[10] R. Reese, M. Thornton, C. Traver, and D. Hemmendinger. Early evaluation for performance enhancement in phased logic. *IEEE Transactions on Computer-Aided Design*, 24(4):532–550, Apr. 2005.

[11] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9(3):189–233, 1996.