Early Evaluation for Performance Enhancement in Phased Logic

Authors:

Robert B. Reese, Mississippi State University (reese@ece.msstate.edu)
Mitchell A. Thornton, Southern Methodist University (mitch@engr.smu.edu)
Cherrice Traver, Union College (traverc@union.edu)
David Hemmendinger, Union College (hemmendd@union.edu)

Early Evaluation for Performance Enhancement in Phased Logic

Abstract

Data-dependent completion time is a well-known advantage of self-timed circuits, one that allows them to operate at average rather than worst-case execution rates. A technique called *early evaluation* that extends this advantage by allowing self-timed modules to produce results before all of their inputs have arrived is described here. The technique can be applied to any combinational function and is integrated into the *phased logic* design methodology that accepts synchronous design entry and produces delay-insensitive self-timed circuits. We describe an algorithm that ensures that the resulting delay-insensitive circuits are safe, and develop a generalized method for inserting early evaluation gates into any phased logic netlist. We give performance results that compare several benchmark circuits, including a 5-stage pipelined CPU and a microprogrammed floating point unit, using early evaluation with non-early evaluation phased logic circuits and with clocked circuits. Simulation results show a clear performance benefit for PL circuits that use early evaluation.

I. Introduction

The ITRS-2001 Roadmap on Design refers throughout to the synchronization and global signaling challenges facing System-On-a-Chip designers up to and beyond 2007. It envisions the common use of hybrid synchronous and asynchronous modules and methods for efficient and predictable implementation of such systems. As a step toward that goal, we describe enhancements to a self-timed design technique that bridges the synchronous and asynchronous worlds. In [1] a two-wire data encoding technique known as level-encoded dual-rail (LEDR) was introduced and applied to self-timed pipelines with compute blocks. Linder and Harden used marked graph theory [6] to generalize this technique in [2][3] to synthesize a safe and live self-timed, delay-insensitive circuit automatically from a netlist of D flip-flops and combinational logic driven by a global clock. They called this self-timed design technique *Phased Logic* (henceforth: *PL*). This paper describes a new technique, *early evaluation* (EE), for improving the performance of PL systems. This technique uses special PL gates that can sometimes evaluate their outputs before all of their inputs have been defined, thus increasing the computation rate. We present a new algorithm to ensure circuit safety with early evaluation gates and a method for automated insertion of early evaluation PL gates into arbitrary netlists [9]. We apply the technique to standard benchmark circuits and to a processor design and compare the results with clocked and non-early evaluation implementations.

In the remainder of this paper, Section II includes an overview of PL, its relationship to marked graphs, and a discussion of the transformation of clocked systems to PL systems. Section III compares the PL approach with similar work. Section IV introduces new work on Phased Logic concerning the definition of an early evaluation gate, a marked graph model of an early evaluation gate, and an algorithm for feedback signal insertion that ensures safety and liveness in the presence of early evaluation. Section V describes a general method for inserting early evaluation PL gates into any PL netlist and two approaches for extracting trigger functions. Section VI discusses other factors that affect the performance of PL systems, and presents a performance-oriented algorithm for feedback insertion. Section VII presents performance results for synthesized PL circuits using early evaluation, and section VIII contains a summary and conclusions.

II. Phased Logic Background

A. Phased Logic

The term "phased logic" was coined by Linder [2] [3] to describe a methodology and logic style that uses an abstraction of signal and gate "phase" to describe circuit behavior. The goal of the work was to produce a digital design methodology that eliminates global clocks, yet keeps the synchronous design paradigm. The methodology allows the use of familiar synchronous design and synthesis tools to produce a clocked netlist of D flip-flops and combinational logic that is then translated to a delay-insensitive PL netlist. A PL netlist consists of PL gates, data signals, and feedback signals. A PL netlist can be considered a Marked Graph (MG) with data tokens flowing throughout the graph. Each data token has a phase that is either *even* or *odd*. The phase is implemented with a LEDR encoding of data signals as illustrated in Figure 1a. Each LEDR signal is composed of a *value* (V) and *timing* (T) wire, and the phase is defined by the parity of the combined dual-rail signal. A PL gate also has an even or odd phase, and it *fires* whenever the phase of all inputs matches the internal gate phase. When the gate fires, the internal phase and the output signal's phase toggles. A sample gate firing is illustrated in Figure 1b,c. A useful property of LEDR signaling is that the V signal always reflects the current value, 0 or 1, of the signal, so this can be used directly in a computation block without decoding. Another useful property is that the phase of a LEDR signal can be changed from even to odd or vice-versa by inverting the T signal. In this way, a PL gate can have both even and odd phase outputs by supplying a version of the LEDR output with the T signal inverted.

Some definitions are required before discussing the MG model of a PL circuit. The following definitions are paraphrased from [5] as they are concise and complete. Expanded definitions can be found in [4].

A Petri net (PN) is a triple PN = {T, P, F} where:

- T is a non-empty finite set of transitions
- P is a non-empty finite set of places, and
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation between transitions and places.

A PN can be represented as a directed bipartite graph, where the arcs represent elements in a flow relation. A PN marking is a function $m : P \rightarrow \{0,1,2...\}$, where *m(p)* is called the number of tokens in *p* under marking *m*. A transition $t \in T$ is *enabled* at a marking *m* if all its predecessor places are marked. An enabled transition *t* may fire, producing a new marking *m'* with one less token in each predecessor place and one more in each successor place (denoted by *m[t > m'*).

A sequence of transitions and intermediate markings $m[t_1 > m_1[t_2 > ...m'$ is called a firing sequence from *m*. The set of markings reachable from marking *m* through a firing sequence is denoted by *[m>*, the *reachability set* of a PN with initial marking *m*. A marking $m \in [m>$ is called a reachable marking.

A PN marking is *live* if for each $m' \in [m$ for each transition *t* there exists a marking $m'' \in [m'>$ that enables *t*. Similarly, a transition t is live if for each $m' \in [m$ there exists a marking $m'' \in [m'>$ that enables *t*. A marked PN is live if its initial marking is live. A marked PN is *k-bounded* if there exists an integer k such that for each place *p*, for each reachable marking *m,* we have *m(p)* $\leq k$. A marked PN is *safe* if it is 1-bounded (this ends the definitions used from [5]).

A PN is a marked graph if every place has exactly one predecessor and one successor. A shorthand graphical notation is usually adopted for MGs in which transitions are the vertices of a graph G and arcs lie between transitions, with the intervening places understood. We use this shorthand notation for MGs in our figures unless explicitly noted otherwise. A Marked Graph G with an initial marking m_0 is denoted (G, m_0). In a MG (G, m_0), a *directed circuit* is a directed path that begins at a transition *t* and ends at the same transition *t*. The sum of the tokens in the set of places contained in C for a marking *m* is the token count of C, designated by *m*(C). Two important theorems [6] about the liveness and safety of MGs are:

Theorem 1. A marked graph (G, m_0) is **live** if and only if for all directed circuits C of G, $m_0(C) > 0$, i.e. m_0 places at least one token on each directed circuit in G.

Theorem 2. A marked graph (G, m_0) is **safe** if and only if every edge of G belongs to some directed circuit C with $m_0(C) \leq 1$. As a corollary, a live marked graph is safe if and only if every edge belongs to a directed circuit C with $m_0(C) = 1$.

If a PL netlist is not live, then signal transitions do not occur (tokens do not circulate), and thus there is no activity in the netlist. A PL netlist requires token circulation for computation, so a dead PL circuit is not useful. If a PL netlist is unsafe, then a PL gate can fire and produce a second output value before a destination gate has consumed the first output value, resulting in incorrect operation.

The MG model for a PL netlist allows representation of gates and LEDR signals with transitions and tokens. Figure 2 shows the token abstraction given an odd or even input signal and an odd or even gate phase using LEDR signaling. An input signal contains a token if the input phase matches the gate phase. When a gate fires, the internal gate phase toggles, thus *consuming* the input tokens. Figure 3 shows that a gate output signal can be viewed as either having a token or not having a token, depending on the destination(s). Because each fanout counts as a separate arc (place) in the MG equivalent, this paper always uses individual signals for fanouts > 1 even though in the physical circuit only two wires are used for an output, regardless of fanout. Figure 4 shows that the initial marking of the MG equivalent of a PL gate is a wiring choice. In the implementation described here, all PL gates are reset to *even* phase at startup. PL gates have both the normal and inverted phase versions of a signal available, accomplished by inverting the internal T signal of the LEDR output. An initial token is placed on an input signal by connecting the signal to the PL gate output whose phase is the same as the internal gate phase.

B. The Transformation Process

The translation of clocked netlists to PL netlists distinguishes between sequential and combinational functions in the clocked netlist. Sequential functions, such as flip-flops, are mapped to *barrier* gates, while combinational functions are mapped to *through* gates.

The terms *barrier* and *through* are used to distinguish these gates for the purpose of initial token marking rules, which are specified later in this section. The logic function of the data values of the LEDR inputs of barrier and through gates are the same as the logic function in the original netlist (i.e., a barrier gate is simply a buffer function in the PL netlist if it does not have any integrated combinational function in the clocked netlist). The translation procedure may need to insert additional signals termed *feedback* signals to make the resulting PL netlist live and safe (a more familiar term is *acknowledgement* signals, but we will use the terminology developed in [2]). In the MG equivalent, a feedback signal is the same as any other directed arc between transitions. However, in the PL netlist, a feedback signal does not have any data associated with it, so a feedback signal from a gate is a single rail signal that is the T wire of the LEDR output of the gate.

The translation algorithm that maps clocked netlists to PL netlists consists of the steps that are outlined below. These steps are illustrated in Figure 5a-d. See [2] and [3] for more details. These rules assume the PL netlist forms a closed system, i.e, that the global reset is the only external input signal. The method for addressing external I/O signals is discussed after the presentation of the translation rules.

Step 1. All sequential functions in the clocked netlist are mapped one-to-one to barrier gates in the PL netlist. All combinational gates are mapped one-to-one to through gates in the PL netlist. LEDR signals connect the PL gates to copy the original topology of the clocked netlist, excluding the clock signal. The initial token marking rules assume that inputs from barrier gates always have an initial token on them. This means that a gate with an input from a barrier gate is connected to the barrier gate output whose phase matches the barrier gate's internal phase. The initial token marking rules require that any non-feedback input signals from through gates do not have an initial token, so these input signals are connected to the through gate output whose phase is opposite the through gate's internal phase. A global reset signal is used to reset all gates to even phase at startup (the even reset choice is arbitrary, it is only necessary that all gates be reset to the same phase value). Figure 5a,b illustrates this step.

Step 2. Extra gates termed *splitter gates* are inserted to break any direct connection between barrier gates. Splitter gates are through gates that implement buffer functions. Splitter gates are required so that the initial token marking rules and feedback insertion rules can result in a live and safe MG. In Figure 5c, through gate *u7* is a splitter gate inserted between barrier gates u5 and u6.

Step 3. The network is traversed, and any signals that are part of a directed circuit C where m_0(C)=1 are marked as *safe* signals. At this point, only output signals from barrier gates are marked with initial tokens, so a signal is safe iff it is in a directed circuit that contains a single barrier gate. It is important to understand that each fanout from an output counts as a separate signal for safety checking. If all signals are safe, then the transformation process is finished, and the PL netlist is live and safe.

Step 4. Single rail signals called *feedbacks* are now added to make the remaining signals safe. A feedback signal is added from the output of a source gate to the input of a destination gate to form a directed circuit C with m_0(C)=1. Any initially unsafe signals contained in this directed circuit are now safe. Any signals made safe by the addition of the feedback are *covered* by that feedback. The directed circuit cannot include a barrier gate unless the barrier gate is either the source or destination of the feedback. Figure 6 summarizes the rules for feedback insertion and initial token marking. A feedback originating from a through gate and terminating on a through gate has an initial token (this marking supplies the token in the directed circuit since the non-feedback output of a through gate does not have an initial token, see figure 6a). A feedback originating from a through gate and terminating on a barrier gate does not have a token (the initial token on the directed circuit is provided by the barrier gate, see Figure 6b). Any feedbacks originating from barrier gates have initial tokens (i.e, all signals originating from barrier gates have initial tokens, see Figure 6c). A feedback cannot both originate from a barrier gate and also terminate on a barrier gate (the directed circuit formed would be unsafe as it would have an initial token count > 1, see Figure 6d). Figure 6e shows how this problem is solved by splitter gate insertion to break barrier-to-barrier gate paths.

Safety is ensured if every signal is part of a directed circuit that contains a single token in the initial marking. To select from multiple feedback signal placement options, a scoring function is used. This scoring function takes into account the number of signals that are covered (made safe), the number of feedback signals previously connected to a gate, and the length of the feedback signal. A general form of the scoring function defined in [2] is given below:

$$\text{score} = S - F/k - p*L \qquad (1)$$

where

- *S* is the number of unsafe signals covered
- *F* is the number of feedbacks present at the destination, and *k* is a user-specified constant. This term is a penalty term that encourages spreading of feedbacks among different nodes. The *k* factor can be used to reduce this penalty if desired. Our mapping code produces PL netlists that use 4-input Muller C-elements [11] for feedback concentration at a node, so *k=4* is used in the netlist mappings results in Section VIII. High-fanout C-elements are built from trees of 4-input C-elements, so spreading fanout among different gates can decrease the delay associated with a feedback input by decreasing the depth of the C-element tree.
- *L* is the number of gate levels between destination and source gates (if the destination is directly connected to the source, F= 1), and *p* is a user-specified constant. This term is a penalty term that favors shorter feedbacks over longer feedbacks as shorter feedbacks can improve the cycle time of the resulting circuit (the effect of feedback length on performance will be discussed more in Section VI). The *p* constant is a weighting factor for this penalty term; our results in Section VII use *p*= 0.1 .

Note that for $k = \infty$ and $p = 0$ the scoring function selects the feedback that covers the largest number of signals. The algorithm [3] for selecting feedbacks using the above scoring function is as follows:

While there are unsafe signals

- For each gate G1 do the following
  - Perform a backward depth-first search starting at G1 along *clear paths*. A *clear path* is one that excludes barrier gates and feedback signals

        except for barrier gates at the beginning or end of the path. This prevents a feedback from forming a directed circuit that includes more than one barrier gate, which would place more than one token on the circuit, forming an unsafe directed circuit.

        - For each gate G2 found in the backward depth-first search, determine the number of signals covered if a feedback signal is added from G1 to G2.
        - Calculate the score for a feedback signal from G1 to G2.
        - If the score is the best seen so far for any G1, save the feedback signal as the current best.

    - Add the best feedback signal seen to the netlist and mark as safe all the signals that are covered by it.

The principle goal of this scoring function is to minimize the number of feedbacks inserted into the PL netlist, and is thus area-oriented. In section VI, we discuss a performance-oriented approach to feedback insertion.

Figure 5 illustrates the mapping of a clocked circuit to a PL circuit with $k = \infty$ and $p = 0$ used in the scoring function. It is evident from Figure 5(d) that multiple solutions exist to feedback insertion for liveness and safety. For example, feedback *f1* was inserted to cover signals *s3* and *s10*. Feedback *f1* originates from gate *u4* and terminates on gate *u2*, where the inputs to *u2* are two gate levels away tracing backwards from *u4* inputs. Feedback *f1* is said to have a *path length* = 2. However, *s3* could also be covered by a feedback from *u4* to *u3* (a path length = 1), and *s10* by a feedback from *u3* to *u2* (a path length = 1). The effect of maximum allowable feedback path length on the performance of PL circuits and on the CPU time required for feedback insertion is discussed in Sections VI and VII, respectively.

The clocked-to-PL translation algorithm has been proven to result in PL circuits that are delay-insensitive, but display the same cyclic, synchronous, deterministic behavior as the clocked netlist [3]. Many example circuits have been translated and compared to their original clocked netlist implementations[7][8][10]. Section IV discusses modifications to the feedback insertion rules to support early evaluation. The safety and liveness of external inputs are handled at the VHDL testbench level during simulation; the PL netlist provides a feedback output to the testbench for each input, and accepts a feedback input from the testbench for each output. The same token marking rules are applied to these feedbacks as are applied in the clocked-to-PL transformation process.

III. Related Work

Several types of delay-insensitive circuits that use LEDR signals have been proposed in the literature. Dean, et al. propose PLA-based, domino logic, and series stack circuit styles in [1]. The sequential behavior of these circuits differs slightly from phased logic gates and they are applied only to pipeline structures. D. L. How describes a self-timed FPGA based upon 3-input lookup tables and LEDR signaling in [13] and uses this cell in the context of Sutherland's micropipelines [16] and self-timed iterative rings [17]. That work does not address the design methodology or EDA tools for the FPGA architecture. McAuley implements wavefront array circuits with a sequential multiplexer cell in [26] and compares their performance with that of clocked systolic arrays. Phased logic differs from all of these approaches by providing a formalization based on graph theory that relates delay-insensitive phased logic circuits to general clocked circuits. Using this formalization, general synchronous circuits can be transformed to phased logic circuits automatically, preserving the synchronous behavior specified by the designer, using commercial EDA tools and a netlist translation tool.

Automated translation of asynchronous designs from synchronous design specifications using commercial synthesis tools has also been proposed in [18]. This asynchronous design methodology, known as Null Convention Logic (NCL), differs from phased logic in several ways. It uses a (NULL/DATA/NULL) signaling convention rather than LEDR and has some delay sensitivity between NCL gates in the final implementation, requiring a final timing analysis. The NCL circuit implementation uses *m*-of-*n* threshold gates. Although both PL and NCL specifications can be written in VHDL RTL, the NCL methodology is restricted to RTL that separates combinational logic and register descriptions.

Performance enhancement techniques for self-timed circuits have been investigated by other researchers under names such as *Eager Evaluation, Speculative Execution*, and *Early Completion* [1][19][20][21][22]. In [1] an "eager" implementation is described for LEDR signaling that is similar in concept to the early evaluation phased logic gates described here. An example of an eager AND gate is given that computes its output value without waiting for both inputs, if one of the inputs has a value of zero. Because the phase of the output is computed from both inputs, for some cases the output phase does not change until both inputs arrive. This dependence limits the speedup to a subset of the potential "eager" transitions that occur for the AND gate. The authors mention the potential problem for eager circuits with feed-forward branches, but they do not provide a solution since the focus of this work was on strict (non-eager) LEDR implementations.

Some speculative completion techniques are applicable only to addition sub-circuits[19]. In [19] a completion detection technique is described for carry-select adders. Another speculative completion technique [20], which uses bundled data and a discrete set of data-dependant matched delays, is a more general design style, but is applied in [20] only to adder circuits. Other performance enhancement techniques that involve completion differ from early evaluation by using optimizations that are unrelated to data dependencies[21][22]. In [21] parallel circuits are used to reduce the delay caused by the NULL part of the DATA/NULL/DATA cycle of NULL Convention Logic. In [22] a technique for detecting completion of NCL circuits at the input of the register, rather than the outputs is described. This optimization is called "early completion" but it is independent of the value of the data inputs. The novelty in the early evaluation approach described here is that it is not limited to addition circuits, it is easily automated, and it takes advantage of data-dependencies in general combinational circuits.

It is also noted that similar methods referred to as "telescopic units" have been employed to speed-up synchronous pipelines [23][24]. The telescopic unit approach considers the early computation of combinational blocks in pipeline stages. The concept of *controlling* values is similar to the concept of early evaluation inputs that compute a trigger function (explained in section IV).

IV. Early Evaluation in Phased Logic

Our extension to the PL methodology is the support for *early evaluation (EE)* within PL netlists. The PL gates discussed in the previous section satisfy the *firing rule,* which states: “A PL gate recalculates its output once all of its inputs have tokens. An input has a token [or: an input has arrived] when the input phase matches the phase of the gate.” An output *update* means that the phase portion of the output is toggled from even-to-odd or vice-versa, and the value portion is updated by the gate compute function. In early evaluation, we divide the set of input signals into two sets, early arriving signals $EI = \{EI_0,EI_1...EI_j\}$ and late arriving signals $LI = \{LI_0,LI_1...LI_k\}$. The firing rule for an early evaluation gate (EEgate) is relaxed so that output update is allowed whenever the early inputs *EI* arrive and the *trigger function Tf= f(EI)*, a boolean function of the value bits of the early arriving signals, evaluates to 1. The trigger function *Tf* is chosen so that the value portion of the gate output is fully determined by the value bits of the early arriving inputs. This technique can provide increased system performance if the signals that form the early fire subset arrive substantially earlier than the remaining signals, as it allows successor gates to begin firing before the remaining inputs arrive. An *early firing* of an EEgate is defined as an output update after all of the early inputs have arrived, but before all of the late inputs have arrived. A *normal firing* of an EEgate is defined as an output update after all inputs $EI \cup LI$ have arrived. An EEgate has two internal gate phases, an *early phase* and a *normal phase,* with a natural extension of the notion of arrival to characterize arrivals of early or late inputs in terms of the corresponding gate phase. Arrival of all early inputs toggles the early phase; arrival of all inputs EI $\cup$ LI toggles the normal phase. A feedback output from an EEgate is updated only after all inputs have arrived (toggling of the normal phase implies toggling of the feedback output phase). Feedback inputs to an EEgate are always inputs to the early phase.

A. Safety and Liveness in Marked Graphs with Early Evaluation

We intend to show that a PL netlist with EEgates can still be modeled as a marked graph, and that only a few simple changes are required to the original feedback insertion rules to produce a MG with EEgates that is live and safe. Figure 7 gives PN models *Eg* and *Ng*, for the early and normal fire behavior, respectively, of an EEgate. Firing of the *Et* transition corresponds to toggling of the early phase, while firing of the *Nt* transition corresponds to toggling of the normal phase. Figure 8 shows *Eg* and *Ng* embedded in PNs $\{Ge, m_0\}$ and $\{Gn, m_0\}$ respectively. It is evident that both $\{Ge, m_0\}$ and $\{Gn, m_0\}$ are marked graphs, and that both are live and safe. *Ge* corresponds to trigger function $Tf = 1$ (the EEgate always fires early), while *Gn* corresponds to trigger function $Tf = 0$ (the EEgate never fires early).

Places $Pint = \{P1, P2\}$ and transitions $Tint = \{Nt, Et, Ot\}$ form the marked graphs of the early (E_g) and normal (N_g) fire subnets. Places $Pext = \{P3, P4, P5, P6, P7\}$, transitions $Text = \{EIt, LIt\}$, and the initial marking m_0 are added to *Eg* and *Ng* to form PNs $\{Ge, m_0\}$ and $\{Gn, m_0\}$, respectively. Initial tokens are placed only within members of *Pext*. In Figure 8, the transition EI_t has an input (place P3) from transition N_t, ensuring that P1 is in a directed circuit C with $m_0(C) = 1$, which prevents a second fire of *Et* from occurring in *Ge* until all inputs have arrived.

Given that an EEgate may fire either early or normally, depending on the trigger function *Tf*, we must express the relationship between these graphs to understand the complete behavior and to show safety and liveness. To explore this point, Figure 9a,b shows the coverability graphs [4] of $\{Ge, m_0\}$ and $\{Gn, m_0\}$. The coverability graph of a PN gives all reachable markings for that PN. Each node in a coverability graph is the marking of the places in the PN. In Figure 9, each node lists places P1 to P7, left to right, represented as a bit string formed from the count of tokens in each place. To emphasize the state of the external places, we assign the identifier for each node to be the decimal representation of the bitstring formed from P3,P4,P5,P6,P7 (shown in bold face in Figure 9). The arcs from each node point to the marking that results when the enabled transition on the arc fires. The set of reachable markings of the set of external places *Pext* is important, as these markings on the input/output arcs of the EEgate define the behavior of the gate to the successor and predecessor gates in the MG model of the PL netlist.

In a safe PN, only '1's and '0's can appear for place values in the coverability graph. Liveness for a PN cannot be determined solely from the coverability graph. However, if a PN is live and is also a MG, then every marking in the coverability graph is live. Therefore, all markings in *Ge* and *Gn* coverability graphs are safe and live. In the coverability graphs in Figure 9c, the dotted arrows show the alternate arcs based on the decision point for the trigger function *Tf* that is evaluated when the early inputs arrive (transition *Et*). These illustrate the reconfiguration from a marking in the early fire coverability graph to the normal fire coverability graph and vice versa. From these coverability graphs, we form the following key observations:

> *The set of reachable markings of Pext for Tf=0 is a subset of its reachable markings for Tf = 1. Furthermore, the alternate arcs for an early or normal fire in Ge lead to a marking in Gn when Tf evaluates to 0, and every marking in Gn is live and safe. The decision point for an early or normal fire in Gn leads to a marking in Ge when Tf evaluates to a 1, and every marking in Ge is live and safe.*

If we restrict consideration to graphs of interest, namely, those that are live and safe, then this observation can be supported by noting that the nodes of the normal-fire coverability graphs form a subset of the nodes of the early-fire graph (since we have identified nodes that differ only in the markings of internal places). Hence, the set of reachable markings for *Pext* is independent of the trigger function *Tf* as long as the *Eg* net is used for the early fire model.

We argue from these observations that the dynamic choice behavior of an EEgate can be modeled as a conditional arc between coverability graphs of MGs. Taking the arc from one graph to another is a *configuration change* in the graph. We define the two possible changes as follows:

- An *early-to-normal configuration change* for an EEgate is a change from the *Eg* graph to the *Ng* graph, triggered by the *Et* transition when *Tf* = 0. The P2 predecessor transition changes from *Et* to *Nt*, and a token appears in P1, but not in P2.

- A *normal-to-early configuration change* for an EEgate is a change from the *Ng* graph to the *Eg* graph, triggered by the *Et* transition when *Tf* =1. The P2 predecessor place changes from *Nt* to *Et*, and a token appears in both P1 and P2.

We will use the above two configuration change definitions in the proof of liveness and safety of EEgates in PL netlists. We begin with a marked graph G composed of EEgates and non-EEgates, where EEgates are represented by *Eg* (*Tf* = 1, always early fire) and non-EEgates by a single transition for all input arcs as used in [2]. Initially G has no feedback arcs, and we produce a live and safe marked graph (G',m_0) by a modified version of the feedback insertion rules presented in Section II. We show that any combination of early-to-normal and normal-to-early configuration changes in the EEgates in G' results in a live and safe MG. We start with three lemmas that characterize the safe and live G'.

Lemma 1: In (G', m_0), each late input LI_i to an EEgate *u* must be in a directed circuit C with m_0(C) = 1 that contains a feedback output arc *fo* from *Nt*.

Proof: Because the EEgate is represented by the early-fire *Eg* net, no late inputs in G are in directed circuits, as no *Nt* transition has output arcs in G. So, a feedback arc *fo* must be added for each late input in order to form a directed circuit for that late input.

Lemma 2: In (G', m_0), at least one early input EI_i to an EEgate *u* must be in a directed circuit C with m_0(C) = 1 that contains a feedback output arc *fo* from *Nt*.

Proof: This is evident from PN *Ge* of Figure 8. If the EI_t transition did not include as an input a feedback output arc, then the firing sequence $EI_t > E_t > O_t > EI_t > E_t$ places two tokens in P1, which is a safety violation.

Lemma 3: In (G', m_0), each output arc O_i from O_t of an EEgate must be in a directed circuit C with m_0(C) = 1 that contains either a feedback input arc fi terminating on *Et* or an early input EI_i.

Proof. This is evident from net *Eg* as there is no path from *Nt* to an output arc O_i from *Ot*. This restricts directed circuits containing an output arc O_i to either contain a feedback input arc or early input arc.

Based on Lemmas 1-3, the modifications to the clocked-to-PL translation rules presented in Section II are:

a) In Step 3, during the marking of initial safe signals, only early inputs are traced through EEgates. This is because *Eg* is used to represent an EEgate, and as such, there is no path from a late input to an output. This means that no late inputs will be marked as initially safe, and must be covered by inserted feedback signals as per Lemma 1. It also means that outputs of EEgates are only marked initially safe if the directed circuit used to mark an EEgate output as initially safe contains an early input (Lemma 3).

b) In Step 3, after the marking of initially safe signals, examine each EEgate u_i. If all early inputs to u_i have been marked as safe, then mark any one early input signal as unsafe. This modification forces a feedback signal to be added in Step 4 to cover this signal as per Lemma 2, preventing a second early fire from occurring until all late inputs for the previous early fire have arrived. Recall that this early input signal can only be initially marked as safe if it is in a directed circuit C containing only one barrier gate. The initial token marking m_0 places initial tokens on the outputs of all barrier gates, so m_0(C)=1, making the signals contained in C safe. Marking one early input signal unsafe within C does not invalidate the safety of the other signals within C as this directed circuit still exists. Thus the feedback to be added in Step 4 only has to cover this early input signal.

c) In Step 4, during the backtracking along clear paths, the path through an EEgate can only include early inputs, as there is no path in *Eg* from the output of an EEgate to a late input.

These changes to the feedback insertion rules will create directed circuits that contain *Pint,* the set of internal places (P1, P2) for the *Eg* nets of all EEgates. These directed circuits will always contain one or more places $Pi \in Pext$, the set of places external to all EEgates *Eg*. The initial token marking rules for MGs without EEgates only place tokens within *Pext* to created directed circuits C_i with m_0(Ci) = 1. The validity of these rules has already been proven in [2]. These same initial token marking rules can be used for G' with no modifications, as places P1 and P2 in *Eg* contain no initial tokens.

With these revised feedback insertion rules, the marked graph (G', m_0) is live and safe if early evaluation gates always fire early (ie, $Tf = 1$). We now show that any combination of early-to-normal or normal-to-early configuration changes still result in a live and safe MG.

Theorem 3: From any marking m_i reachable from (G', m_0), allow a single EEgate u_i to perform an early-to-normal configuration change. The resulting graph (G'', m_0') is live and safe.

Proof: The only directed circuits C with $m_\mathrm{i}(C) = 1$ affected by the early-to-normal configuration change are the ones containing an output arc from *Ot*, as the predecessor transition to P2 is now *Nt*. All these directed circuits now contain P1 as a result of the configuration change. P1 has a token from the firing of *Et*, so these directed circuits are live, $m_\mathrm{i}(C) > 0$. When *Et* fired, the only arcs in these directed circuits that could have contained a token are the arcs incident on *Et*. The firing of *Et* consumed these tokens, so the token count of these directed circuits remain unchanged, at $m_\mathrm{i}(C) = 1$.

Theorem 4: From any marking m_i reachable from (G', m_0), allow any set of EEgates $U=\{u_0,u_1,u_2...u_n\}$ to perform early-to-normal configuration changes. The resulting graph (G'', m_0') is live and safe.

Proof: The only directed circuits C with $m_i(C) = 1$ affected by the early-to-normal configuration changes are the ones containing an output arc from *Ot* within a gate $u_i \in U$. Pick any directed circuit Ci affected by an early-to-normal configuration change. Because Ci was live and safe in (G', m_i), the *Et* transition change within Ci that triggers the early-to-normal configuration change is the only fireable transition on this directed circuit. Thus, any of the other early-to-normal configurations changes do not affect this directed circuit, so by the same reasoning in the proof of Theorem 6, Ci is live and safe. Because all directed circuits Ci can be affected by only one of the early-to-normal configuration changes in U, then all directed circuits with $m_i(C) = 1$ affected by these configuration changes are live and safe.

At this point, we have proven that any combination of early-to-normal configuration changes from (G', m_i) result in a marked graph (G'', m_0') that is live and safe. As long as no normal-to-early configuration changes occur in G'', the graph is live and safe as all markings reachable from (G'', m_0') are live and safe. Also, any combinations of additional early-to-normal configuration changes from any (G'', m_i') resulting in (G''', m_0'') are also live and safe by Theorems 3 and 4. We will now consider normal-to-early configurations changes from G''.

Theorem 5: From any marking m_i' reachable from (G'', m_0'), allow a single EEgate u_i to perform a normal-to-early configuration change. The resulting graph (G''', m_0'') is live and safe.

Proof: The only directed circuits C with m_i'(C) = 1 affected by the normal-to-early configuration change are the ones containing an output arc from *Ot*, as the predecessor transition to P2 is now *Et*. All these directed circuits now no longer contain P1 as a result of the configuration change, but they do still contain P2. The firing of *Et* that causes the normal-to-early configuration change places a token in P2, so these directed circuits are live, m_i'(C) > 0. As these directed circuits have m_i'(C) = 1 at the time of *Et* firing, the only arcs in these directed circuits that could have contained a token are the arcs incident on *Et*. The firing of *Et* consumed these tokens, so the token count of these directed circuits remain unchanged, at m_i'(C) = 1.

Theorem 6: From any marking m_i' reachable from (G'', m_0'), allow any set of EEgates U={u_0,u_1,u_2...u_n} to perform normal-to-early configuration changes. The resulting graph (G''', m_0'') is live and safe.

Proof: The only directed circuits C with m_i'(C) = 1 affected by the normal-to-early configuration changes are the ones containing an output arc from *Ot* within a gate $u_i \in$ U. Pick any directed circuit Ci affected by an normal-to-early configuration change. Because Ci was live and safe in (G'', m_i'), the *Et* transition change within Ci that triggers the normal-to-early configuration change is the only fireable transition on this directed circuit. Thus, any of the other normal-to-early configurations changes do not affect this directed circuit, so by the same reasoning in the proof of Theorem 5, Ci is live and safe. Because all directed circuits Ci can be affected by only one of the normal-to-early configuration changes in U, then all directed circuits with m_i'(C) = 1 affected by these configuration changes are live and safe.

Theorems 3-6 prove that any combination of early-to-normal and normal-to-early configuration changes in the live and safe marked graph (G', m_0) produce another live and safe MG. The key point in inserting feedbacks in a PL netlist to provide liveness/safety in the presence of EEgates is to make the graph live and safe assuming that the EEgates always fire early; then any combination of early and normal fires are live and safe.

One outcome of using the same initial token marking rules for a PL netlist with or without EEgates, is that an EEgate can either be a barrier gate or a through gate, as early evaluation is independent of initial token marking. Of course, it is meaningless for a barrier gate to be an EEgate if it implements a buffer function, as there is only one input to the gate. However, there are two situations where a barrier gate with early evaluation capability could be useful:

(a) the DFFs in the original clocked netlist have multiple inputs that implement a combinational function (this accommodates the common situation in ASIC libraries that embed logic within DFFs).

(b) If a DFF does not have embedded combinational logic, then after Step 1 of the translation process and before splitter gates are inserted, a netlist transformation is performed in which DFFs are absorbed into the combinational gate that supplies their data input. Absorption of a DFF into a combinational gate can be done if the DFF is the unique fanout of that combinational gate. This can reduce the critical path of the circuit, improving performance. However, this can also create a direct DFF-to-DFF path if there was only one combinational gate between DFFs, and this performance gain is lost when splitter gates are inserted in Step 2 of the clocked-to-PL transformation process.

V. Generalized Insertion of Early Evaluation Gates

The previous section shows that the insertion of early evaluation gates can be accomplished without making phased logic circuits unsafe. However, early evaluation gates will be more expensive to implement than standard PL gates, so it is important to use them only when a significant performance advantage can be obtained. An early evaluation gate implements two logic functions. The *master* function, M(I), is the original function mapped to the gate from its corresponding clocked netlist element. The *trigger* function, $T(I_T)$, has only a subset of the inputs of the master function, $I_T \subset I$. The trigger function is true for those values of I_T such that $M(I_T) = M(I)$. A technique for finding potential trigger functions for these logic functions, selecting the optimal trigger functions, and then selecting which netlist elements should be implemented with early evaluation gates is described in this section.

Two approaches are taken for the generation of trigger functions; an exhaustive search for small functions and a heuristic method for larger functions. We will first assume 4-input logic functions. This is a reasonable upper bound for standard logic cells and a common size for field-programmable gate array lookup tables. With this assumption, all 14 possible subfunctions with three or fewer inputs can be evaluated as trigger functions and a merit function can be applied to choose the trigger function with the best characteristics. A function can be a trigger function if it is true for at least some cases when the master function value is independent of the non-trigger inputs. Candidate trigger functions are computed by processing the cube list representation of the f^{ON} and f^{OFF} functions for the master function, M(I).

As an example, consider the truth table for a carry-out function of a full adder cell, $M(a,b,c) = c(a+b)+ab$, as shown in the Master column of Table 1. Since this function depends on three input signals, a search for the trigger function would consist of generating all candidate functions with support sets of $\{a\}$, $\{b\}$, $\{c\}$, $\{a,b\}$, $\{a,c\}$ and $\{b,c\}$. In the Trigger column of Table 1 a potential trigger function is shown, $T(a,b) = ab+a'b'$, that is based on the support set $\{a,b\}$.

Table 1: Truth Tables for Master and Trigger Functions

a	*b*	*c*	*Master*	*Trigger*
0	0	0	0	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

Each time the trigger function is true, the master gate can evaluate even if the input signal *c* has not arrived since its value is a don't-care in these cases.

The best trigger function may be determined by means of a merit function that measures the degree of coverage that the trigger function provides in relation to the master Boolean function. The c*overage* is the number of times a master function may early evaluate divided by 2^n where n is the number of elements in the set *I* (i.e. $|I|$). The higher this percentage is, the more often early evaluation can occur. Table 2 shows the cube representation of the master full-adder carry-out function along with the computed coverage. Since 2 cubes in Table 2 depend only upon master inputs *a* and *b* and each of those cubes cover 4 of the 8 possible outputs of the master function, a coverage of 50% is computed for the trigger function $f_{trig}=ab+a'b'$. The trigger function corresponds to the cube list given by f^{ON}_{trig}={00-,11-}.

Table 2: Determination of Candidate Trigger Functions

Master Cube	Master Outputs	{*a*,*b*} Coverage	Trigger Function
00-	0	2	1
010	0	0	0
100	0	0	0
11-	1	2	1
1-1	1	0	0
-11	1	0	1

The merit function is also weighted by the relative arrival times of the input signals to the master and candidate trigger PL gates. This is necessary since, unlike the case of the adder circuit, a potential trigger function with large coverage may depend on slowly arriving signals and hence provide less effective speed-up than a trigger function with less coverage but depending on faster arriving inputs. The arrival times are computed by finding the maximum path length in terms of PL gates from the primary circuit inputs or from barrier gate outputs to PL gate inputs. The merit function is given in Equation (2).

$$Merit = \%Coverage \times \frac{M_{\max}}{T_{\max}} \tag{2}$$

M_{max} and T_{max} are the maximum delay of the input signals to the master and trigger PL functions, respectively. This merit function works well if data movement in the forward direction from external inputs or barrier gates limits performance, as is the case in a clocked system. However, if performance is limited at a gate by feedback arrival, then insertion of an EEgate will not help. To account for this, the effect on cycle time of the PL netlist by the proposed EEgate insertion could be incorporated into the merit function. However, this is a difficult issue, as typically the insertion of just one early evaluation gate does not improve cycle time. Instead, groups of gates (i.e, one gate for each bit of a datapath) have to be inserted before improvement in cycle time is seen. Also, how to determine cycle time in a netlist that has data-dependent cycle times caused by the presence of EEgates is a difficult problem. Future work is planned on evaluating different merit functions for EEgate insertion.

This technique presented here generalizes EE to work for any arbitrary master function since candidate trigger functions are automatically extracted based on the structure of the master function. If the merit function is low for all trigger functions, the use of early evaluation gates may not be worth the additional overhead, and in those cases a standard PL gate can be used instead. The designer can select a threshold value for the merit function, below which an early evaluation gate is not inserted.

As the size of the master functions increase in terms of the variable support set, it becomes impractical to search over all possible candidate functions since a total of 2^n-2 functions must be evaluated. To avoid this exponentially large exhaustive search, a heuristic method using *Binary Decision Diagrams* (BDDs) can be used [14]. A BDD is a directed acyclic graph that represents a fully specified switching function $f : \mathbf{B}^n \rightarrow \mathbf{B}$. All vertices in a BDD are annotated with either a Boolean constant or a dependent variable. Those vertices annotated with a constant are terminal nodes with the remainder of the vertices referred to as non-terminal nodes. All non-terminal vertices have two exiting directed arcs while terminal vertices have no exiting edges. One of the non-terminal vertices is denoted as the initial node of the BDD and it has no edges pointing to it. These graphs are ordered in the sense that for any given path from an initial to a terminal vertex, all intermediate vertices encountered are annotated with variables that may occur only once during the traversal and always in the same order regardless of the path. BDDs are also reduced in the sense that all isomorphic subgraphs are shared and that no redundant vertices exist. Additionally the edge set of a BDD also carries annotations in the form of Boolean constants. Each non-terminal vertex has two exiting edges labeled with a 1 or a 0. A switching function may be evaluated for a particular variable assignment by traversing a path from the initial to a root node and following the directed edges that correspond to the polarity of each variable for the particular assignment. The annotation of the root node is then the evaluation of the function. As an example, Figure 10 contains a diagram representing a BDD for the master function given in Table 1 with a variable ordering of a, b, c respectively.

It is a property of BDDs that all paths from the initial to a root vertex represent disjoint cubes in the on- or off-set of the function being represented. By definition, a trigger function is one that is composed of a set of cubes in the on-set that represent cubes in either the on- or off-set of the master function and which depend on fewer variables than the master function. Such cubes are easily identified from a BDD representation of the master function by extracting those that correspond to paths from the initial to a terminal node such that the paths do not include all variables. In the example master function in

Table 1, a trigger function can be extracted depending only on variables *a* and *b*. In the corresponding BDD shown in Figure 10a, note that variables *a* and *b* are ordered first and that complete paths exist by following the 0-edge of variables *a* and *b* yielding the cube as the on-set of the trigger function. These complete paths are illustrated in the BDD in Figure 10b where two paths are shown by the dashed ovals that correspond to the cubes {*abc*}={00-,11-}. The technique for extracting a trigger function is then to construct a BDD with all variables in the set I_T to be grouped together and appear in the BDD structure closest to the initial node with the remaining variables in *I* to appear closer to the root nodes. Next, the BDD is traversed and all distinct paths containing variables in $\mathrm{I_T}$ are chosen as cubes in the on-set of the trigger function whether those paths terminate at terminal nodes annotated with 0 or 1.

An additional constraint in determining the trigger function is the desire to have variables in the support set that are early arriving. This information is obtained through a timing analysis of the PL circuit prior to trigger function extraction and is used to determine candidate variable orderings for the BDD. Those variables that are earliest in arrival are ordered first in the BDD. Those variables with equal arrival times are grouped together in the BDD. This method has been used to extract trigger functions from candidate master functions that depend on as many as 34 variables in less than 1 ms of computer runtime.

VI. Other Performance Considerations in PL Circuits

A. Performance of Timed Marked Graphs

An elementary directed circuit in a MG is a directed circuit that contains no other circuits. A timed marked graph assigns delays to each transition in the MG; the MG model of a PL netlist is a timed MG. Assuming fixed delays for each transition, a lower bound on the cycle time of a timed MG can be found by computing the average cycle time D(C*i*) of each elementary directed circuit C_i of G in isolation, by summing the firing times of the nodes in C_i and dividing by the number of tokens present on C_i. The lower bound on cycle time is then *max*{D(C*i*)} [31] . Figure 11 shows a simple PL netlist that is an unbalanced two-stage pipeline, where B1/B2 are barrier gates and T1-T4 are through gates. Recall that barrier gates and through gates are DFFs and combinational gates in the original clocked system. The elementary directed circuit in Figure 11a formed by B1>T1>T2>T3>B2>T4>B1 contains two tokens and forms the directed circuit with the maximum average cycle time. Assuming each node has a delay of 10 time units, then the average firing time of this directed circuit is 60/2 = 30 time units. Analysis shows that the firing pattern at each node is 20, 40, 20, 40, etc. for an average delay of 30 time units. In a clocked system, the cycle time of this system would be 40 time units if B1, B2 were DFFs, and T1-T4 were combinational blocks, as the cycle time of the clocked system is set by the maximum delay between DFFs.

The cycle time of a PL netlist without early evaluation gates is fixed and is independent of input vector value. The cycle time of a PL netlist with early evaluation gates can vary from cycle to cycle based upon input vector values because the early or normal firing of an early evaluation gate is data-dependent. As such, the cycle times of PL circuits with early evaluation gates presented in Section VII are average cycle times measured over a range of different input vectors. There is a one-to-one correspondence between a firing cycle in a PL netlist and a clock cycle in the original clocked system. Thus, the cycle time of the PL netlist is analogous to the clock period of a clocked netlist.

A register-to-register path in a clocked circuit corresponds to a barrier gate to barrier gate path in the PL circuit. In a clocked circuit, only one computation can be in progress on the register-to-register path at any given time during a clock cycle, unless an asynchronous technique such as wave pipelining is used. However, in a PL circuit, it is possible to have more than one token in flight between barrier gates, which would correspond to more than one computation in progress between the barrier gates. This allows PL system performance to be somewhat tolerant of unequal delays between barrier gates. This is seen Figure 11a as the delays across the two stages of the pipeline in the PL system are averaged because the PL system can support two tokens in flight between B1 and B2; something that cannot occur in the clocked system.

B. Feedback insertion and cycle time

In a PL netlist, directed circuits occur either naturally via paths from the output of a barrier gate back to its input, or are created by the addition of feedback. Figure 11b is a minimal feedback solution to the circuit of Figure 11a. The directed circuit formed by B1>T1>T2>T3>B1 is now the circuit with the maximum average cycle time. This circuit has one token, with average cycle time of 40/1 = 40 time units. The firing pattern at each gate is now 40,40,40...etc. Analysis shows that the firing of gate B1 is caused by token arrival on the feedback input from T3, not by token arrival from T4. The firing of gate B1 is limited by token movement in the backwards direction along feedbacks, not by token movement in the forward direction along non-feedback inputs. This illustrates how feedback insertion can have an adverse effect on the performance of the final PL circuit. Long feedbacks (feedbacks that skip over multiple gate lengths) generate directed circuits with many gates, which can limit the performance of the PL netlist. To remove feedback path length as a performance factor, all circuit examples in Section VII use a feedback path length of 1 unless explicitly stated otherwise. This maximizes the number of feedbacks, but also maximizes circuit performance for our current mapping approach.

The original scoring function for feedback insertion in [2] is area-oriented, not performance-oriented. To create a performance-oriented feedback insertion algorithm, the effect on cycle time on the resulting PL netlist must be considered when feedback is inserted in the netlist. Unfortunately, identification of the elementary directed circuit with the maximum cycle time is an NP-complete problem [33]. A lower bound on the average cycle time of a timed marked graph can be computed in polynomial time using a linear programming approach [31]. Another approach for computing the cycle time of a marked graph involves the solution of a set of linear equations that use a special max-plus algebra [32]. The applicability of these approaches for computing cycle time in PL netlists with EEgates is questionable given that the number of nodes can be large and the token flow data-dependent.

We have implemented a performance-oriented algorithm for feedback insertion that uses a simulation-based approach for computing cycle time. We feel that a simulation approach is required because of the data-dependent cycle times in PL netlists with EEgates. The cycle time of the PL netlist is calculated via a timed MG simulator that is integrated into the mapping software. The MG simulator currently assumes that all EEgates always fire early; however we plan on extending it to use firing patterns for EEgates captured from an external gate level simulator. Our performance-oriented algorithm uses the MG simulator to iteratively identify gates fired by feedback arrival, and targets those feedbacks that have the most waiting time between the last non-feedback arrival and the gate firing. These feedbacks are removed, and the resulting unsafe signals are covered by feedbacks restricted to half of the maximum path length of the deleted feedbacks. The goal is to have those gates previously fired by token movement in the backward direction (feedback arrival), to now be fired by token movement in the forward direction (non-feedback arrival). The performance-oriented algorithm for feedback insertion is:

1. Identify a target average cycle time *Tavg* for the final PL netlist. We currently find *Tavg* by creating a PL netlist where all feedbacks are restricted to path length =1. This gives the minimum achievable cycle time for this PL netlist using our current mapping approach.
2. Create a PL netlist using the area-oriented scoring function for feedback with no restriction on feedback path length. This creates a PL netlist with the minimum

number of feedbacks using our current search technique for feedback insertion. This netlist is used as the starting point for the optimization process.

3. Run the MG simulator until a stable average cycle time is reached where the average cycle time is computed as a running average over the last four cycles. This usually takes less than 10 simulation cycles. During the simulation, track gates that are fired by feedback arrival, and not by non-feedback input arrival. Time spent waiting for feedback is wasted time. We want gates to be triggered by data movement in the forward direction, as in the clocked system. If the target cycle time has been reached, then the mapping is finished.
4. Traverse the list of gates *Gl* fired by feedback arrival, and identify the gates *Glmax* that have the largest delta time between the last non-feedback arrival and the feedback arrival that fires the gate. For each of these gates, identify the feedback input *fi* that triggered the firing of the gate. If the path length of the feedback is the largest so far, record as *Pmax*. Place the feedback *fi* in a list *Fl*. After all gates *Glmax* have been processed, if *Pmax* = 1 then the mapping is finished as the length of the late arriving feedbacks cannot be further reduced.
5. For each feedback $fi \in Fl$, remove *fi* from the PL netlist and mark the signals covered by *fi* as unsafe.
6. Insert feedback to cover all unsafe signals created in step 5 using the area-oriented scoring function, restricting feedback length to *Pmax/2*. Go to step 3.

This algorithm is a greedy heuristic as no candidate feedbacks in step 4 are rejected. Results from using this approach are given in Section VII.

C. Slack Matching Buffers

Another method of improving the performance of the PL netlist involves adding extra buffers called slack-matching buffers [30] in order to alter the average cycle time of directed circuits within the netlist. Circuit A in Figure 12 is a modified version of the original two-stage pipeline found in Figure 11. The dashed lines are feedbacks of length 1. The directed circuit formed by B2>T1>T2>T3>B2 is now the circuit with the maximum average cycle time. This circuit has one token, with average cycle time of 40/1 = 40 time units. However, by adding the extra buffer T5 shown in Circuit B, the directed circuit with the maximum average firing time becomes B1>T1>T2>B3>T4, the same directed circuit found in Figure 11a. The firing pattern of each gate is changed to a repeating pattern of 40,20,40,20, etc, for an average cycle time of 30 time units, an improvement of 10 time units. In the examples in Section VII, any slack matching that has been done to improve performance is explicitly stated and is accomplished manually. An area of future work is to incorporate automatic insertion of slack matching buffers into our mapping tool.

VII. Results

A. The Clocked-to-PL mapping environment

We evaluated the effectiveness of early evaluation for gates of 4 inputs or less, by synthesizing several benchmark circuits both with and without the use of early evaluation gates. The benchmark circuits were the *International Test Conference* 1999 (ITC99) suite [14], a 5-stage pipelined MIPS CPU[28], and the picoJava-II™ Floating Point Unit

[29]. Circuit sizes ranged from 4 gates to over 8500 gates and a variety of circuit structures are included. The circuits were specified in RTL-level VHDL and Verilog formats, and each test case was synthesized using the Synopsys© Design Compiler tool with a minimum delay constraint to an EDIF netlist of D-flip-flips and 4-input lookup tables (LUT4s). Early evaluation functions are extracted from this netlist using the techniques described in section V and are written to a separate file. The PL mapping program reads both the EDIF netlist and early evaluation function file, maps this to a netlist of normal and early-evaluation PL gates, inserts feedback signals using the algorithm described in Sections II and IV, and produces a VHDL netlist that is simulated via the Mentor Graphics Modelsim VHDL simulator. The complete mapping flow is shown in Figure 13. The logic synthesis via Design Compiler from RTL to a gate-level netlist has no knowledge of early-evaluation or PL in general. Logic synthesis constraints used to produce faster or slower clocked circuits will in general also produce faster or slower PL circuits. An area of future work is changing logic synthesis algorithms to be early-evaluation and PL aware, so that the resulting netlist has more opportunity for speedup.

A 4-input lookup table was chosen as the basic combinational element because one application of PL technology would be as the basis for a family of self-timed Field Programmable Gate Arrays (FPGAs). Circuit details for LUT4-based PL gates with early evaluation capability are presented in [8]. A summary of PL gate operation is presented here for completeness. Each PL gate has five inputs; four LEDR inputs for the logic function and one single-rail feedback input. The internal logic function is stored in a

LUT4, whose output is latched upon gate firing to produce a LEDR output. A Muller C-element is used to detect input arrival, and the state of the C-element determines the gate phase and the value of the feedback output. Unused logic inputs are either used as extra feedback inputs, or tied to the feedback output of the gate. Trees of four-input C-elements are used to concentrate feedback inputs to a gate if necessary. To remove technology dependence, all circuit performance results are normalized to LUT4 delays (one LUT4 delay = 1.0). A PL gate has delay equal to 1.4 because of the overhead of the output latch after the internal LUT4 that contains the gate function. A PL gate with early evaluation is implemented using two normal PL gates; one implementing the trigger function T(EIi) and early transition Et, as defined in section IV, and one implementing the normal function and transition Nt. Because of the extra complexity of this gating, a PL gate with early evaluation has delay of 1.6 LUT4 delays. The four-input Muller C-elements used for feedback concentration have delay of 0.6. Note that a PL system has a 40% gate-level delay penalty with respect to the clocked netlist because of the output latch latency. A PL gate is similar to a Sutherland micropipeline block [16], and a common feature of micropipelines is that output latch latency is in the critical path of the circuit. The performance boost from early evaluation can be used to overcome this latency penalty.

C. *International Test Conference* 1999 (ITC99) Benchmarks

Table 3 gives the performance results from mapping the *International Test Conference* 1999 (ITC99) benchmarks to PL netlist implementations. For the 15 benchmarks presented in Table 3, 11 cases demonstrated a performance improvement after insertion of early evaluation gates. Furthermore 11 of the 15 cases had less than the expected 40%

performance degradation when compared to their clock counterparts, and three benchmarks had better performance than the clocked netlists. In these results, EE circuitry was added to all PL gates where a speedup was possible. No slack matching buffers were added to any of these circuits.

Table 3 contains columns representing the description of the benchmark circuit, total PL gates required without EE, total PL gates required with EE, the percent area increase in terms of additional gates when the EE algorithm is applied, the cycle time of the clocked design (longest register-to-register path), the average cycle times of the non-EE and EE PL netlists, the percent performance increase for non-EE versus EE, and the percent performance increase for clocked versus the PL EE netlist. Synopsys© Design Compiler was used to find the longest register-to-register path in the clocked design, with the delay of a DFF plus setup time equal to 1.0 LUT4 delay.

Because a PL gate with early evaluation has a longer delay than a non-EE PL gate, some benchmarks suffered a slight degradation in overall delay values when the EE algorithm was applied. Overall, the EE algorithm resulted in a speedup of the PL netlist for most of the benchmarks. Not surprisingly, those benchmarks with significant amounts of arithmetic circuitry benefited more from the EE algorithm since arithmetic circuits are frequently composed of addition circuits where EE techniques are known to perform well[20]. In the more complex examples (Viper and 80386 processors), the speedup gained from early evaluation was able to overcome the 40% gate-level penalty of PL gates and enable the PL system to have equivalent performance to the clocked system.

Note that in the Viper and 80386 benchmarks, the PL netlist without EE had lower cycle time than the original clocked netlists; this performance boost is from delay averaging of unequal paths lengths between DFFs.

Table 3: Experimental Results Comparing the Use of EE in PL Synthesis

	Area			Performance				
Description	PL Gates (no EE)	EE Gates	% Area Increase	Clk Cycle time	PL Cycle (no EE)	PL Cycle (w. EE)	% Cycle (noee vs ee)	% Cycle (clk vs ee)
FSM that compares serial flows	30	35	17%	7	9.2	8	13%	-14%
FSM that recognizes BCD numbers	8	8	0%	2	4.0	4	0%	-100%
Resource arbiter	96	111	16%	7	7.8	9	-10%	-23%
Computer min and max	341	456	34%	12	19.6	16	18%	-33%
Elaborate contents of memory	304	410	35%	16	14.8	14	8%	15%
Interrupt handler	18	20	11%	5	7.0	8	-14%	-60%
Count points on a straight line	261	350	34%	10	12.6	12	8%	-16%
Find inclusions in sequences	99	114	15%	11	11.2	12	-4%	-5%
Serial to serial converter	93	112	20%	7	8.6	8	2%	-20%
Voting system	121	157	30%	9	12.6	12	6%	-31%
Scramble string with a cipher	379	585	54%	13	15.4	14	9%	-8%
1-player games (guess a sequence)	584	766	31%	12	16.8	18	-6%	-48%
Interface to meteo sensors	170	195	15%	7	9.2	8	15%	-11%
Viper processor (subset)	3409	5789	70%	40	37.8	27	29%	33%
80386 processor (subset	5122	8035	57%	42	34.4	28	19%	34%

D. *A 5-Stage Pipeline CPU*

A deeper exploration of the benefits of early evaluation was done via the mapping of a 5-stage pipeline CPU [28][34] that implements a subset of the MIPS ISA. Figure 14 shows a simplified diagram of the pipeline structure. The circled multiplexers indicate architectural-level application of early evaluation. One location for application of early evaluation is in the forwarding path from the ALU to decode stage; if this forwarding path is not needed for the current instruction, then this multiplexer fires early allowing the decode stage to begin execution before the ALU result is ready. Similarly, if the next PC value did not require the computed PC from the branch logic, then this multiplexer fires early. The multiplexer that interfaced the external input databus to the rest of the CPU was also replaced with EE gates. If the instruction was not a load word (*lw*), this allowed the rest of the CPU to proceed without having to wait for the memory interface to fire. These EE gates were inserted manually into the architecture in order to observe their effect upon performance. Other CPU versions had additional EE gates that were inserted automatically into the netlist via the procedure in Section V. A limitation on our tool that performs automated insertion of EEgates is that we currently have no method for specifying delays of external inputs, so the early evaluation multiplexers for the external memory inputs have to be inserted manually; we plan on correcting this limitation in the future. Table 4 gives the different versions of the CPU that were generated to explore the benefits of early evaluation.

Table 4: MIPS CPU Implementations

PL Versions (clocked version had 6134 gates, delay path of 24)	Gates	% Extra gates
a) No EE gates, no slack matching buffering	6231	0%
b) Manually inserted EE gates, no slack matching buffering	6424	3.1%
c) Version (b) + slack matching buffering	6453	3.6%
d) Version (c) + automated insertion of EE gates, with trigger gates chosen by a cost function that weights signal arrival times with a trigger function coverage of 50% or better	7207	15.7%
e) Version (c) + automated insertion of EE gates on all LUTs with input signal arrival time differences of one LUT delay or better	8252	32.4%

Version (e) inserted early evaluation gates wherever a possible speedup could be obtained; the same as was done for the ITC benchmarks. Version (d) utilized user-specified constraints to the early evaluation trigger function extraction algorithm which limited trigger function choice based on signal arrival times and function coverage. Note that version (e) requires almost twice as many extra gates as version (d).

Five benchmark programs were used for performance measurement: (a) fibonacci (fib), a value of 7 was used, (b) bubblesort, an array size of 10 was used, (c) crc, calculate a CRC table with 256 entries, (d) sieve – find prime numbers, stopping point set to 40 (e) matrix transpose - a 20x30 matrix was used. Table 5 shows the PL cycle times in LUT4 delays for the various versions running the CRC benchmark. The register-to-register path of the clocked design was 24 LUT4 delays. The column marked as "CRC" is average cycle time of the PL system for the execution of the CRC benchmark using code produced by the *gcc* compiler with a compile optimization level of "-O". The small difference between versions (d) and (e) indicates the importance of limiting trigger function extraction, as there is a point of diminishing returns for performance gained versus gates added. As opposed to the ITC benchmarks, every version with early evaluation was less than the clocked cycle time of 24 LUT4 delays.

Table 5: Performance for CRC Benchmark

Version	CRC	% improvement	CRC (RO)	% improvement
(a)	25.2		25.2	
(b)	22.2	12%	21.6	14%
(c)	18.9	25%	17.8	29%
(d)	17.3	31%	16.1	36%
(e)	17.6	30%	16.3	35%

The column marked as "CRC (RO)" is the CRC benchmark with the instruction stream as produced by the *gcc* compiler reordered so as to reduce ALU operand forwarding. For example, a typical code segment produced by *gcc* is shown below:

```
addi   r4,r4,1
slti   r2, r2, 8
bne    r2, r0, L10
```

ALU forwarding is required for the *bne* instruction because *r2* is a destination in the *slti* instruction, and a source in the *bne* instruction. However, the instructions can be reordered as shown below:

```
slti   r2, r2, 8
addi   r4,r4,1
bne    r2, r0, L10
```

Functionally, the two code streams are equivalent, but the second code stream does not require ALU forwarding for the *bne* instruction, which increases the number of early evaluation firings and hence the performance. Instruction reordering was done manually by examining the assembly code of the critical loops. Table 6 gives the performance for all benchmarks using reordered instruction streams. The CRC benchmark had the fastest average cycle time as it had the highest percentage of logical operations whose cycle time benefited the most from the insertion of early evaluation gates. The Fib benchmark was fast as it had the lowest amount of ALU operand forwarding. Instruction reordering is an example of an application-level modification to take advantage of the speedup offered by early evaluation.

Table 6: Performance for all MIPs Benchmarks

	Fib		Bubble		CRC		Sieve		Tpose		Avg
Ver a.	25.2		25.2		25.2		25.2		25.2		
Ver b.	21.8	13%	22.0	13%	21.6	14%	22.2	12%	22.0	13%	13%
Ver c.	17.8	29%	18.8	26%	17.8	29%	19.0	24%	18.5	27%	27%
Ver d.	16.6	34%	17.5	30%	16.1	36%	17.5	30%	16.9	33%	33%
Ver e.	16.8	33%	17.8	29%	16.3	35%	17.5	30%	16.9	33%	32%

E. *picoJava-II™ Floating Point Unit*

The floating-point unit in the picoJavaII CPU is a microcoded engine with a 32-bit datapath that performs single and double precision floating point operations in IEEE 754 format. The microcode is stored in two 160 x 54 bit ROMs. The FPU is available from Sun Microsystems as Verilog RTL. Before synthesis, the Verilog RTL was restructured to place the microcode ROMs external to the hierarchy so that a PL interface wrapper could be placed around them. The Verilog RTL was then synthesized to a netlist of DFFs and LUT4s and mapped to a PL implementation. The cycle time of the clocked netlist was 40 LUT4 delays and contained 8559 gates. Table 7 shows the PL cycle times for the individual FPU instructions. The EE version of the FPU was mapped using the same trigger function constraint options as for version (d) of the 5-stage pipelined CPU. Note that all of the instruction cycle times of the EE version of the FPU are lower than the clocked version. The increase in gates from the clocked version to the PL (none-EE) version is due to the insertion of through gates (splitter gates) between directly connected DFFs by the mapping program as mentioned in Section II. No slack matching buffers were added in the mapping of the FPU.

Table 7: FPU Instruction Cycle Times

Clocked version	8559 gates, cycle time = 40		
PL (None-EE)	8573 PLgates		
PL (EE)	12130 PLgates (42% increase over non-EE version)		
	Cycle Time (LUT4 delays)		
Instructions	No EE	EE	%change
fadd/fsub	50.4	31.6	37%
fcmpg	50.4	31.8	37%
fcmpl	50.4	31.8	37%
fdiv	50.4	35	31%
fmul	50.4	31.6	37%
frem	50.4	34.4	32%
f2d	50.4	31.6	37%
f2i	50.4	31.6	37%
f2l	50.4	31.4	38%
i2f	50.4	31.4	38%
l2f	50.4	31.4	38%
dadd/dsub	50.4	31.4	38%
dcmpg	50.4	31.2	38%
dcmpl	50.4	31.4	38%
ddiv	50.4	36.2	28%
dmul	50.4	31.4	38%
drem	50.4	33.2	34%
d2f	50.4	31.4	38%
d2i	50.4	31.4	38%
d2l	50.4	31.4	38%
i2d	50.4	31.6	37%
l2f	50.4	31.4	38%
Average	50.4	32.1	36%

F. *Mapping Performance*

Table 8 gives netlist results using different mapping constraints for the MIPS, CPU, ITC B14 and ITC B15 benchmarks. These tests were run on a 2.0 GHz P4 computer with 1 GB of main memory under RedHat Linux. Column meaning from left to right is:

- *FB Alg (PO Iter):* This indicates if the area-oriented (AO) or performance-oriented (PO) feedback insertion algorithm was used. For the PO algorithm, the number in parenthesis is the number of iterations of steps 3-6 required to meet the target performance.

- *PLgates:* number of PLgates in the equivalent MG for the PL netlist.
- *Signals*: number of signals in the equivalent MG for the PL netlist; each fanout from a gate is a separate signal.
- *Unsafe*: number of signals unsafe after initial safe net marking.
- *Max FB Len (Actual):* This is restriction on the maximum feedback path length. The number in parenthesis is the actual maximum feedback length used in the netlist.
- *Fbacks*: number of feedbacks inserted
- *%chg*: percent change in number of feedbacks required from base case of FBlen=1.
- *Map Time (s)*: mapping time in seconds measured via Unix *time* utility.
- *EE trigger extract*: early evaluation trigger extraction time
- *MG Sim Cycle Time*: The cycle time in LUT4 delays as reported by the MG simulator contained within the mapping tool; this does not simulate the datapath.
- *VHDL Sim Cycle Time*: The cycle time in LUT4 delays as reported by the VHDL simulation of the PL netlist.
- *%chg*: percent change in the VHDL simulation cycle time from the base case of feedback length = 1.

Table 8: Netlist results under different mapping constraints

	FB Alg (PO Iter)	PLgates	Signals	Unsafe	Max FB Length (Actual)	Fbacks	%chg	Map Time (s)	EE Trigger Extract (s)	MG Sim Cycle TIme	VHDL Sim Cycle Time	%chg
MIPS	ao	6231	17170	12672	1	12572		6.7		25.2	25.2	
	ao				max(20)	6200	51%	52		32.2	31.8	-26%
	ao[2]				max(14)	7763	38%	30		25.2	25.2	0%
	po (11)				max(11)	6320	50%	68		25.2	25.2	0%
MIPS/ee	ao	7207	17614	14726	1	14626		7.7	4.7	11.8	16.6	
	ao				max(11)	9531	35%	25		16.5	21.4	-29%
	po (88)				max(6)	12012	18%	189		11.8	19.2	-16%
FPU	ao	8573	27511	18197	1	17971		91		50.4	50.4	
	ao				max(24)	8227	54%	2062		50.4	50.4	0%
	ao[2]				max (31)	8159	55%	388		50.4	50.4	0%
FPU/ee	ao	11550	27511	25478	1	25350		17	8.6	22.4	31.6	
	ao				max(11)	13174	48%	103		23	31.6	0%
B14	ao	3409	11653	9392	1	9306		30		37.4	37.4	
	ao				max(19)	3177	66%	969		55.2	55.2	-48%
	ao[2]				max(24)	3211	65%	263		54.6	54.6	-46%
	po[2] (212)				max(16)	6344	32%	492		37.4	37.4	0%
B14/ee	ao	5789	11653	10961	1	10890		9	1.6	13.2	27	
	ao				max(10)	6774	38%	52		20.2	27	0%
B15a	ao			11409	1	11303		955		33.9	34.4	
	ao				max(22)	5321	53%	2889		56.6	56.6	-65%
	ao[2]				max(25)	5454	52%	1263		52.4	52.4	-52%
	po[2](162)				max(17)	7026	38%	1445		33.9	34.4	0%
B15b	ao[1]	5122	16456	16456	1	16350		10		33.9	34.4	
	ao[1,2]				max(25)	7137	56%	280		54.4	54.4	-58%
	po[1,2](232)				max(17)	9591	41%	591		33.9	34.4	0%
B15/ee	ao	8035	16456	14603		14584		11	1.9	16	28	
	ao				max(13)	10455	28%	41		22.7	34.8	-24%
	po (65)				max(8)	11490	21%	156		16	30.2	-8%

[1] initial safe net marking not performed, [2]: initial feedback destinations restricted

Some general observations based on Table 8 are:

1. The lowest cycle times were obtained for FBlen = 1. In most cases, having an unrestricted feedback length resulted in longer cycle times, but not for the FPU,

FPU/ee or B14/ee cases. For these cases, the PO feedback insertion was not performed, as the AO feedback insertion did not affect performance.

2. The cycle time reported by the MG simulator is in good agreement with the cycle time reported by the VHDL simulator for the non-EE cases. However, the cycle time reported by the MG simulator is very optimistic for EE cases since it assumes that EEgates always perform an early fire.
3. The PO feedback insertion algorithm always met the target cycle time, with a reduction in the number of feedbacks required when compared to the FBlen= 1 case. However, meeting the target MG cycle time for the EE netlist cases does not guarantee that the VHDL cycle time will be the same as the case for FBlen =1 because of the optimistic cycle time of the MG simulator. The PO feedback insertion algorithm created netlists with lower performance for the MIPS/ee and B15/ee cases because the iteration terminates when the target performance is reached – more late-arriving feedbacks can still be removed, but these do not affect the MG simulation time, while they do affect the VHDL simulation time. This implies that a more accurate cycle time simulation is needed for netlists with EEgates. This could be achieved by using the actual EEgate firing sequences captured from the VHDL simulator. A firing sequence of early/normal fires for an EEgate does not change as long as the netlist is live and safe and thus will not change for different feedback arrangements. A firing sequence would only have to be captured once, using a representative set of input vectors.
4. The B15a, B15b cases illustrate that initial safe net marking can dominate the CPU time required for mapping if the combinational network has high fanout,

increasing the number of paths to be traced. The B15b case disabled initial safe net marking, resulting in the insertion of more feedbacks.

5. High fan-in within a combinational network can cause the back tracing for feedback destination gates to dominate the mapping CPU time. To reduce execution time for complex netlists when there is no restriction on feedback path length, feedback insertion was divided into two passes. The first pass restricted starting points for back tracking to barrier gates and early evaluation gates. Barrier gates were chosen as the starting point because only output signals from barrier gates cannot be covered by feedback originating from a barrier gate. EEgates were chosen because the late arriving inputs must be covered by feedback from the EEgate. First pass feedback insertion was terminated when the number of signals covered by the next best choice feedback dropped below a user-specified threshold *Smin*. The second pass feedback insertion algorithm then proceeded normally, with feedback path length restricted to *Smin*. This two-pass approach was tested with for the non-early evaluation cases in Table 8 with *Smin* = 5. Except for the MIPS, the restricted-search feedback insertion algorithm significantly reduced the mapping time and came very close to matching the number of feedbacks inserted by the original search algorithm. For the FPU, the restricted search algorithm actually beat the original search algorithm by a small amount. The two-pass feedback insertion algorithm was not used with the early evaluation benchmark cases because feedback destination candidates are already limited as back tracing through EEgates is restricted to early inputs.

We feel that the results justify the use of a simulation-based approach for a performance-driven feedback insertion for the non-EEgate test cases. Improvement of the cycle time estimation is needed with this approach for early-evaluation PL netlists, perhaps by including EEgate firing sequences captured from a gate level simulation.

VIII. Conclusion

A technique called early evaluation has been described for improving the performance of phased logic circuits. An extension of a previously published translation algorithm has been shown by means of marked-graph theory to result in live and safe phased logic circuits. The inclusion of this technique in the phased logic design flow allows a designer to specify a circuit in VHDL or Verilog, synthesize it to a clocked netlist, translate it to a PL netlist, and then make tradeoffs between increased area and performance through the automated insertion of early evaluation gates. This technique has been shown to improve the performance of several benchmark circuits of various architectural types, including a pipelined integer CPU and a microcoded floating point unit.

REFERENCES

[1] M.E. Dean, T.E. Williams, and D.L. Dill, "Efficient Self-Timing with Level-Encoded 2-Phase Dual-Rail (LEDR)," *Advanced Research in VLSI*, 1991.
[2] Daniel H. Linder and James C. Harden, "Phased Logic: Supporting the Synchronous Design Paradigm with Delay-insensitive Circuitry," *IEEE Transactions on Computers*, vol 45, no 9, September 1996.
[3] Daniel H. Linder, *Phased Logic: A Design Methodology for Delay-Insensitive Synchronous Circuitry*, PhD thesis, Mississippi State University, 1994.
[4] T. Murata, "Petri Nets: Properties, Analysis and Applications," *IEEE Proceedings*, vol. 77, pp. 541-580, April 1989.

[5] A.Yakovlev, M. Kishinevskh, A. Kondratyev, L. Lavagno, “On the models for asynchronous circuit behavior with OR causality”, Technical Report Series No. 463, Computing Science, University of Newcastle upon Tyne, November 1993.

[6] F. Commoner, A. W. Holt, S. Even, A. Pneueli, "Marked Directed Graphs", *J. Computer and System Sciences*, vol. 5, pp. 511-523, 1971.

[7] R. B. Reese, M. A. Thornton, and C. Traver, “ Arithmetic Logic Circuits using Self-timed Bit-Level Dataflow and Early Evaluation”, *Proceedings of the 2001 Conference on Computer Design*, pp 18-23, September 2001.

[8] C. Traver, R. B. Reese, M. A. Thornton, “Cell Designs for Self-timed FPGAs”, *Proceedings of the 2001 ASIC/SOC Conference*, pp 175-179, September 2001.

[9] M.A. Thornton, K. Fazel, R.B. Reese, and C. Traver, “Generalized Early Evaluation in Self-Timed Circuits”, *Proceedings of DATE 2002*, Paris France, March 4-8, 2002.

[10] R. Reese, and C. Traver, "Synthesis and Simulation of Phased Logic Systems", Technical Report MSSU-COE-ERC-00-09, MSU/NSF Engineering Research Center, June 2000. Presented at International Workshop on Logic Synthesis (IWLS 2000), Dana Point, CA, June 2, 2000.

[11] D.E. Muller and W. S. Bartky, "A Theory of Asynchronous Circuits", in *Proc. Int. Symp. on Theory of Switching*, vol. 29, pp.204-243, 1959.

[12] Tzyh-Yung Wuu and Sarma B. K. Vrudhula, "A Design of a Fast and Area Efficient Multi-Input Muller C-element", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol 1, no. 2, June 1993.

[13] Dana L. How, “A Self Clocked FPGA for General Purpose Logic Emulation”, in proceedings of *IEEE 1996 Custom Integrated Circuits Conference*, pp. 148-151, 1996.

[14] Electronic CAD and Reliability Group, ITC99 Benchmark Set-Politecnio di Torino, http://www.cad.polito.it/tools/itc99.html, 1999.

[15] R.E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, *IEEE Transaction on Computers*, vol. C, no. 35, pp. 677-691, 1986.

[16] I. Sutherland, “Micropipelines”, *Communications of the ACM*, vol 32, no. 6, pp. 720-738, June 1989.

[17] M.R. Greenstreet, T.E. Williams, and J. Staunstrup, "Self-Timed Iteration", *VLSI '87*, C. H. Sequin (Ed.), Elsevier Science Publishers, pp. 309-322, 1988.

[18] Michiel Ligthart, Karl Fant, Ross Smith, Alexander Taubin, Alex Kondratyev, "Asynchronous Design Using Commercial HDL Synthesis Tools", In *Sixth Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async 2000)*, Eilat, Israel, April 2000.

[19] A. DeGloria and M. Olivera, “Completion-detecting Carry Select Addition”, *IEE Proceedings-Computer and Digital Techniques*, vol. 147, no. 2, pp. 93- 100, 2000.

[20] S. M. Nowick, K. Y. Yun, P. A. Beerel and A. E. Dooply, “Speculative Completion for the Design of High-performance Asynchronous Dynamic Adders”, *Proceedings of Advanced Research in Asynchronous Circuits and Systems*, pp. 210-223, 1997.

[21] S. C. Smith, R. F. DeMara, J. S. Yuan, M. Hagedorn and D. Ferguson, “Speedup of Delay-insensitive Digital Systems Using NULL Cycle Reduction”, *Notes of the International Workshop on Logic and Synthesis*, pp. 185-189, 2001.

[22] S. C. Smith, “Speedup of Self-Timed Systems Using Early Completion”, *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2002),* Pittsburgh, Pennsylvania, pp. 107-113, 2002.

[23] L. Benini, E. Macii and M. Poncino, “Telescopic Units: Increasing the Average Throughput of Pipelined Designs by Adaptive Latency Control”, *Proceedings of the Design Automation Conference*, pp. 22-27, 1997.

[24] L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso and M. Poncino, “Automatic Synthesis of Large Telescopic Units Based on Near-Minimum Timed Supersetting”, *IEEE Transactions on Computers*, vol. 48, no. 8, pp. 769-779, 1999.

[25] L. Y. Rosenblum, A. V. Yakovlev, “Signal graphs: from self-timed to timed ones”, *Proceedings of International Workshop on Timed Petri Nets,* IEEE Computer Society Press, Torino, Italy, pp. 199-207, 1985.

[26] Anthony J. McAuley, “Four State Asynchronous Architectures”, *IEEE Transactions on Computers*, vol. 41, no. 2, February 1992.

[27] James L. Peterson, “Petri Nets”, *Computing Surveys*, vol. 9, no. 3, September 1977.

[28] Anders Wallander, “A VHDL Implementation of a MIPS”, Project Report, Dept. of Computer Science and Electrical Engineering, Luleă University of Technology, http://www.ludd.luth.se/~walle/projects/myrisc.

[29] “picoJava-II™ Microarchitecture Guide”, Sun Microsystems, Inc, Part No.:960-1160-11, March 1999, http://www.sun.com/processors/communitysource/index.html.

[30] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, Tak Kwan Lee, “The Design of an Asynchronous MIPS R3000 Microprocessor”, *Proceedings of the 17th Conference on Advanced Research in VLSI*, pp. 164-181.

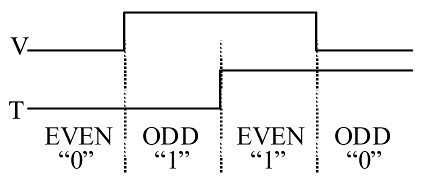
[31] J. Campos, G. Chiola, J. Colom, M. Silva, “Properties and Performance Bounds for Timed Marked Graphs”, IEEE Transactions. Circuits and Systems, vol. 39, pp. 386-401, May 1992.

[32] F. Baccelli, G. Cohen, G. Olsder, J. Quadrat, *Synchronization and Linearity: Algebra for Discrete Event Systems*. New York: John Wiley & Sons, 1992.

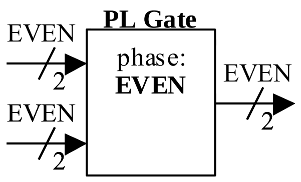
[33] J. L. Chen, *Analysis of Distributed System Software in Maintenance Phase Based on Timed Petri Net Model.* Ph.D. dissertation, Northwestern University, 1987.

[34] R. Reese, M. Thornton and C. Traver, "A Fine-grained Phased Logic CPU", *IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2003)*, Tampa, Florida, February, 2003, pp. 70-79.

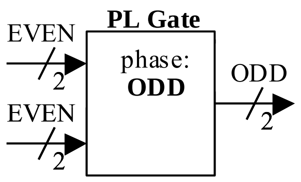
Figures



a. LEDR encoding

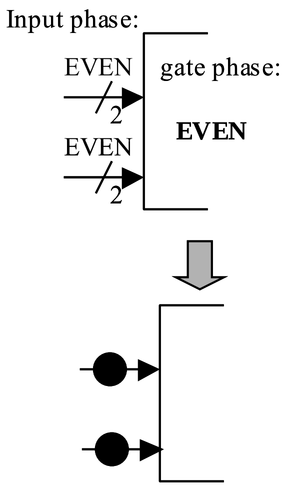


b. Ready to fire

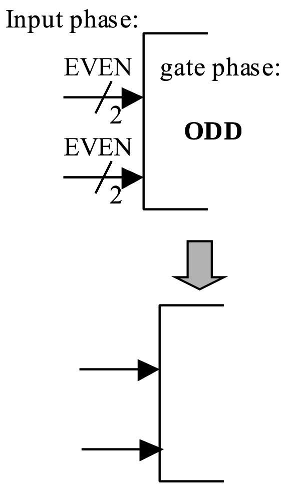


c. After gate fires

Figure 1. LEDR Encoding and PL Gate Firing



a. Ready to fire



b. After firing

Figure 2. Token abstraction (input signals)

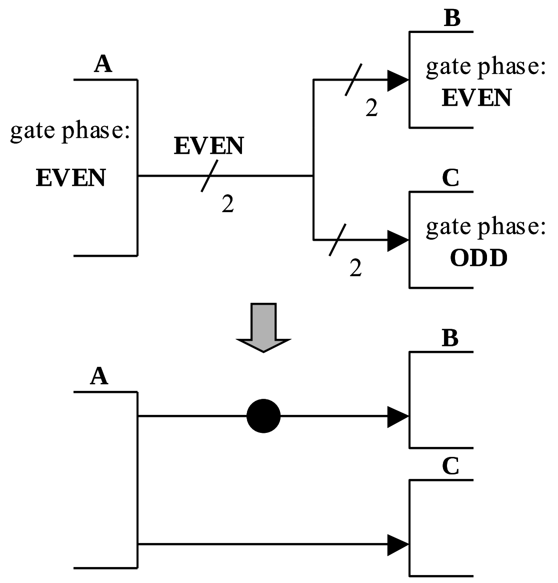


Figure 3. Token abstraction (outputs)

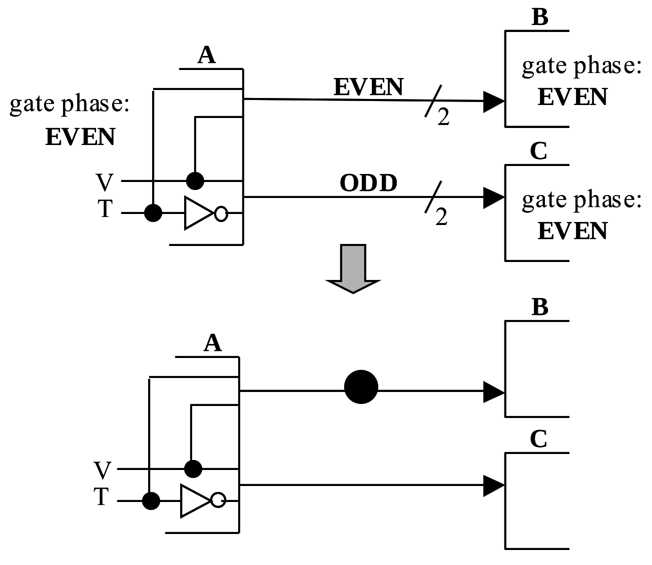
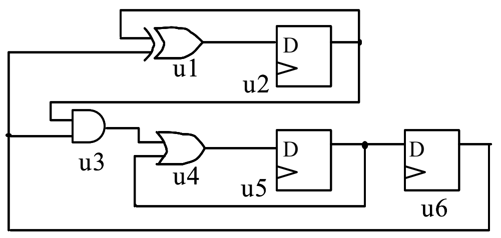
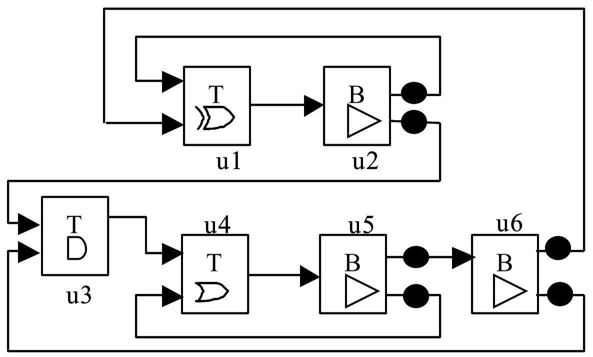


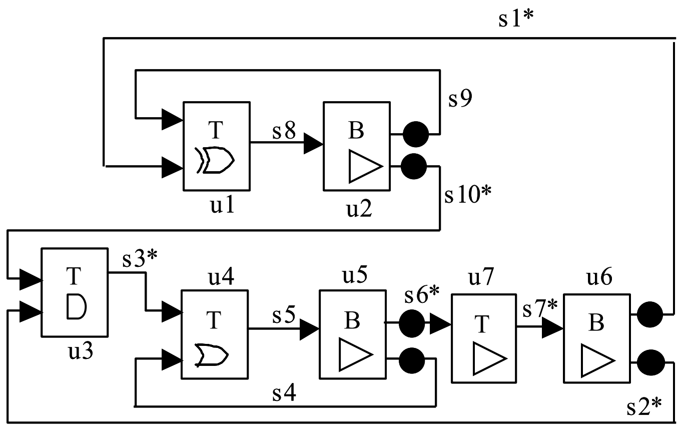
Figure 4. Initial token marking is a wiring choice



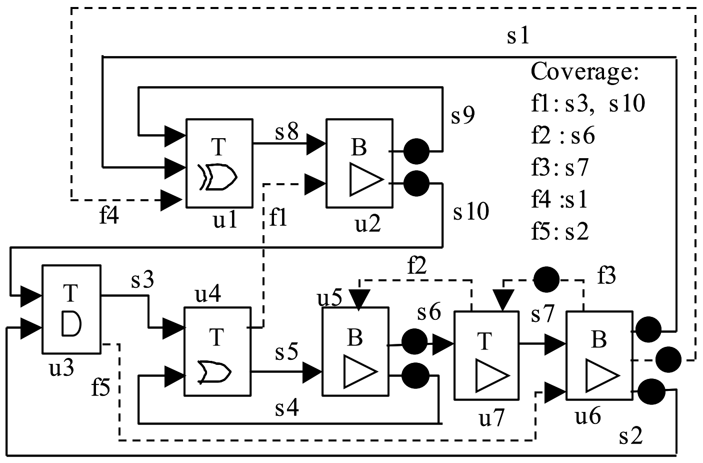
a) Clocked circuit



b) PL circuit before splitter gate insertion



c) PL circuit after splitter gate insertion, s*i** indicates an unsafe signal in initial marking



d) After feedback insertion, circuit is live and safe.

Figure 5: Example translation

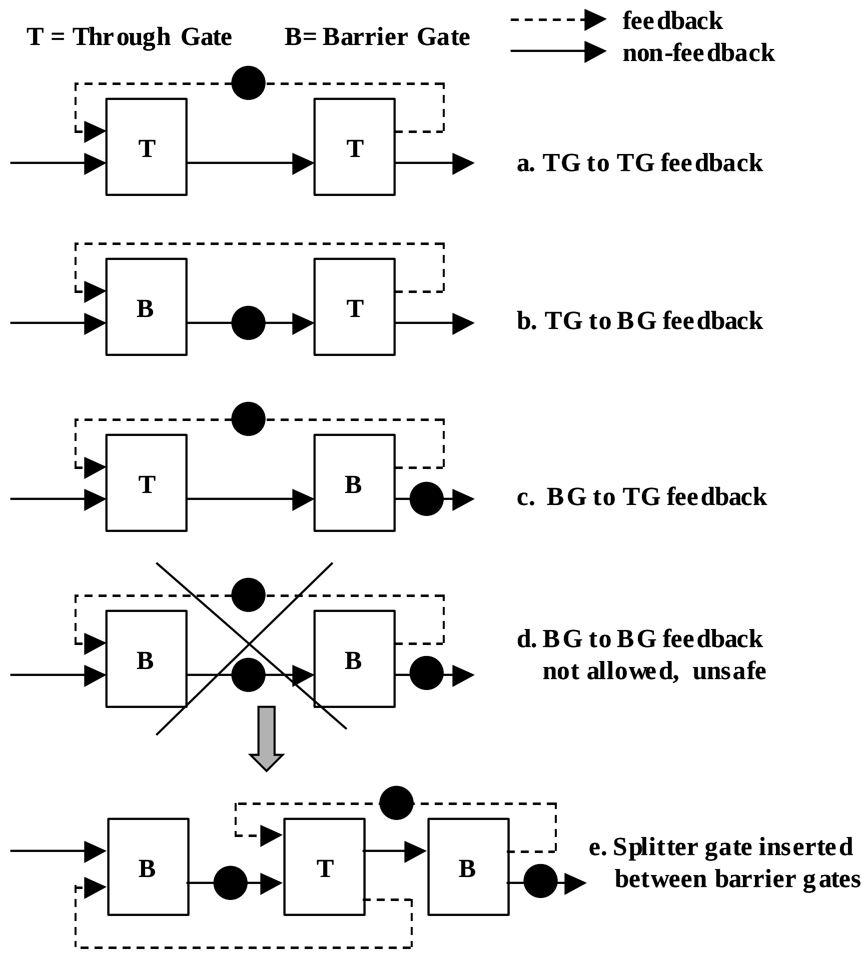
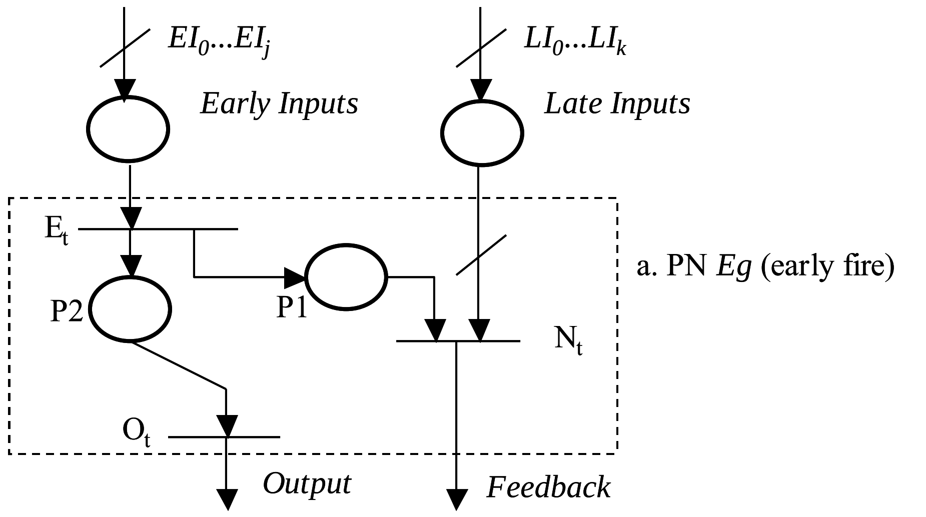


Figure 6: Token marking, feedback insertion, splitter gate insertion rules



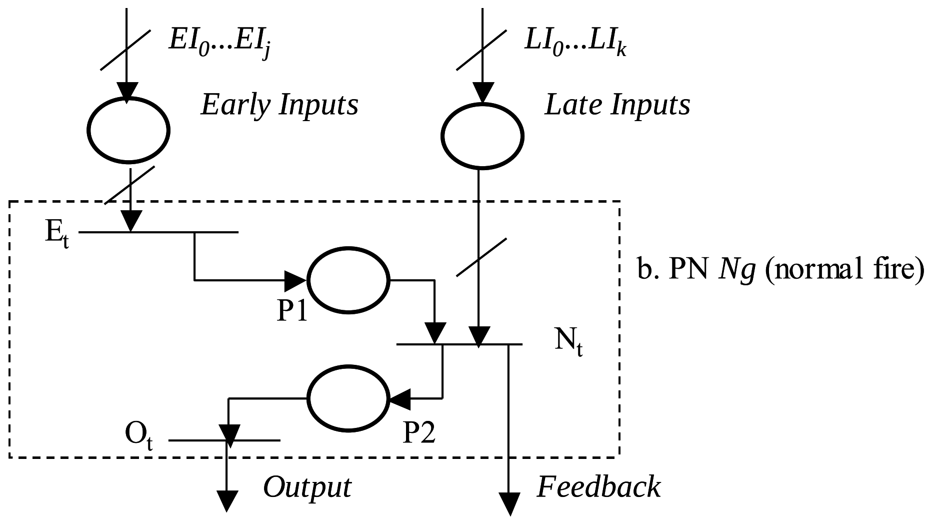


Figure 7. Petri Net models of EEgate early and normal fire behaviors

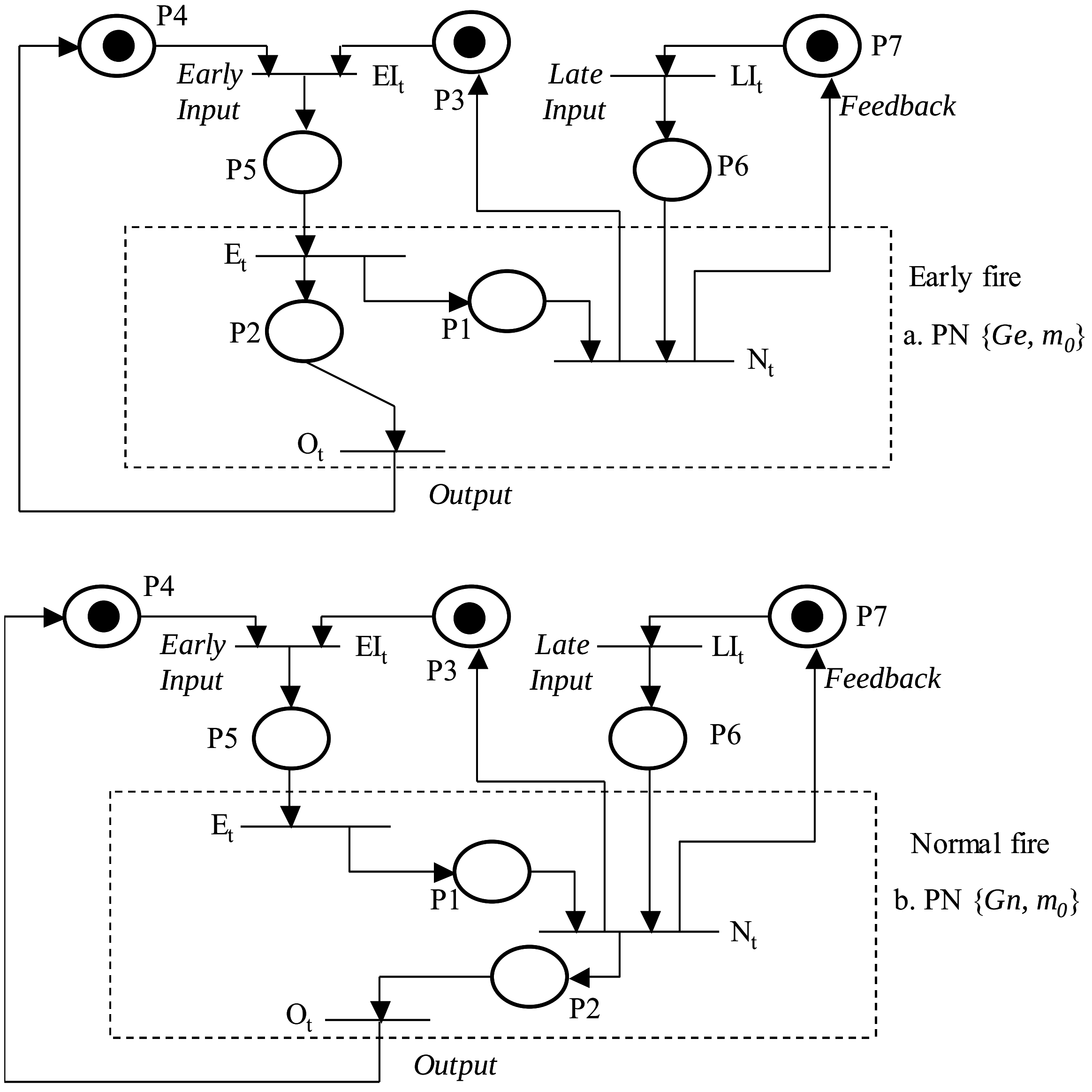
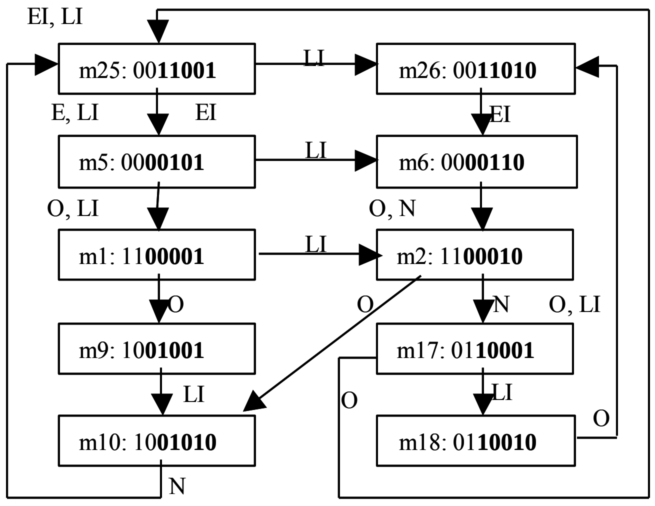
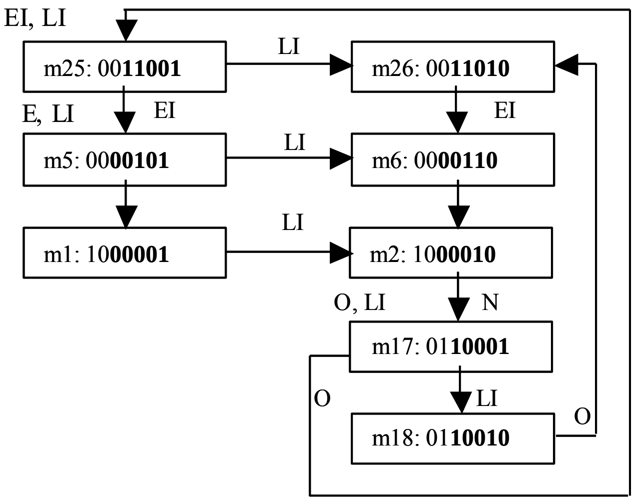


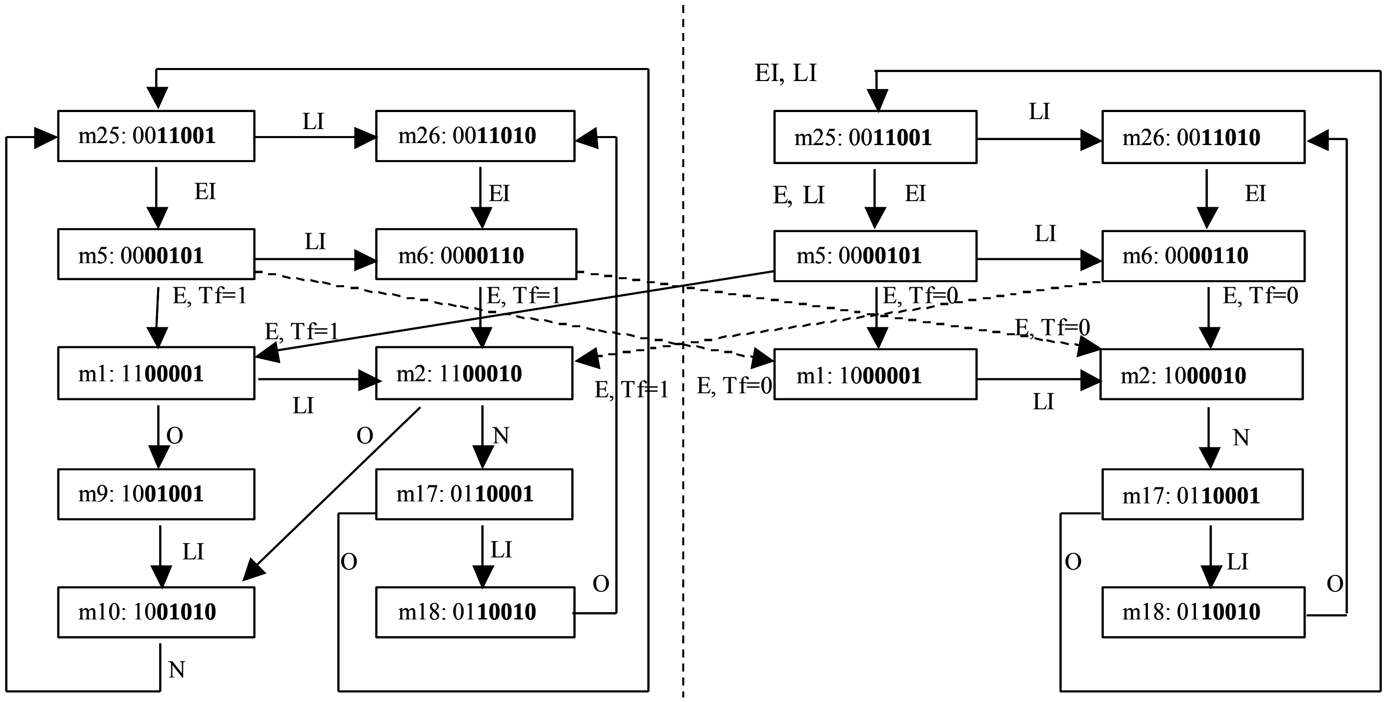
Figure 8. *Eg* and *Ng* within live/safe MGs



a) Early Fire



b) Normal Fire



c) Configuration change arcs

Figure 9. Coverability graphs for EEgate early fire and normal fire marked graph models

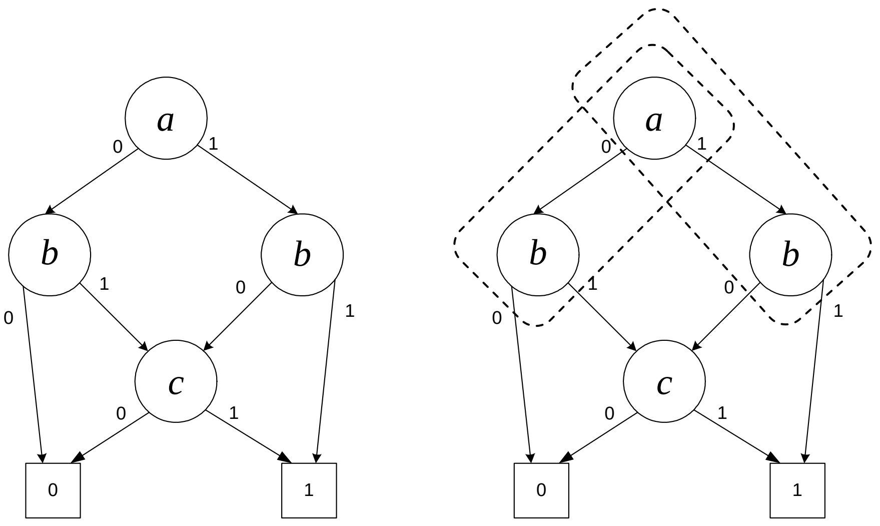
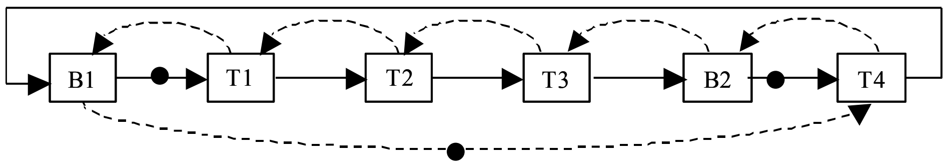
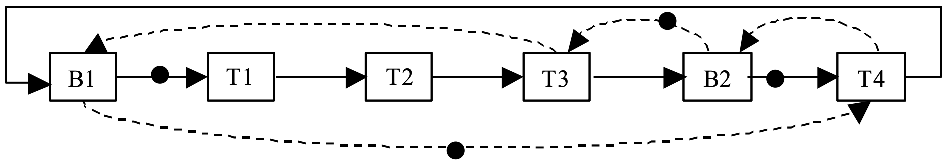


Figure 10. Example BDD for Master Function in Table 1



a. Block delay = 10 units, Firing pattern at B1 = 20,40,20,40... for average = 30



b. Block delay = 10 units, Firing pattern at B1 = 40, 40, 40... for average = 40

Figure 11. Performance Example

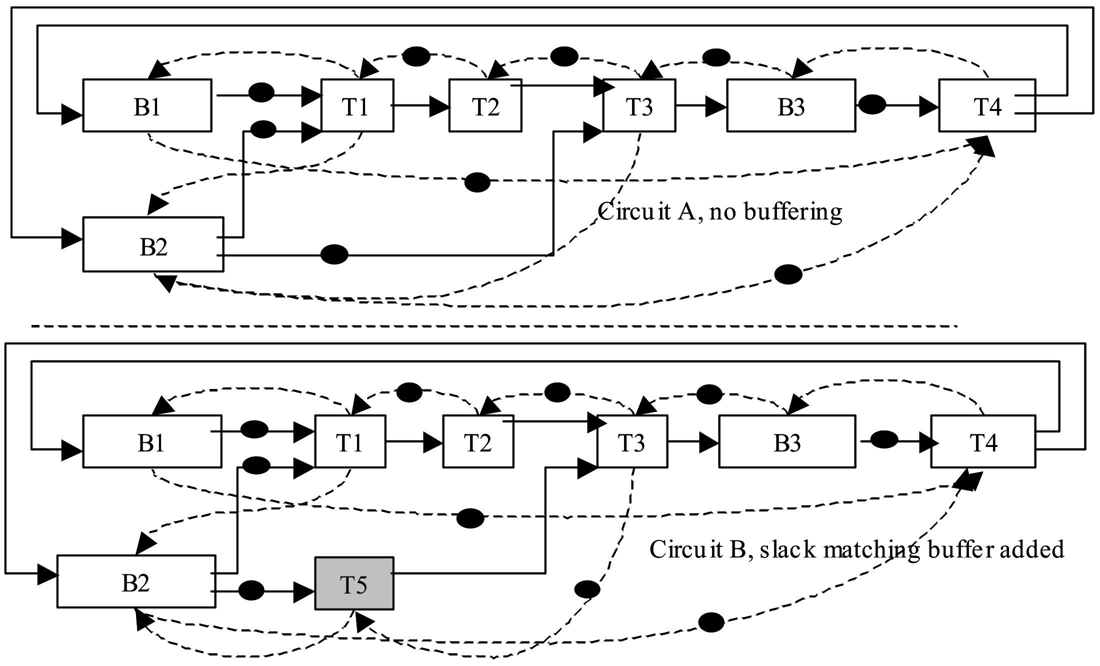


Figure 12. Slack Matching Buffer Insertion

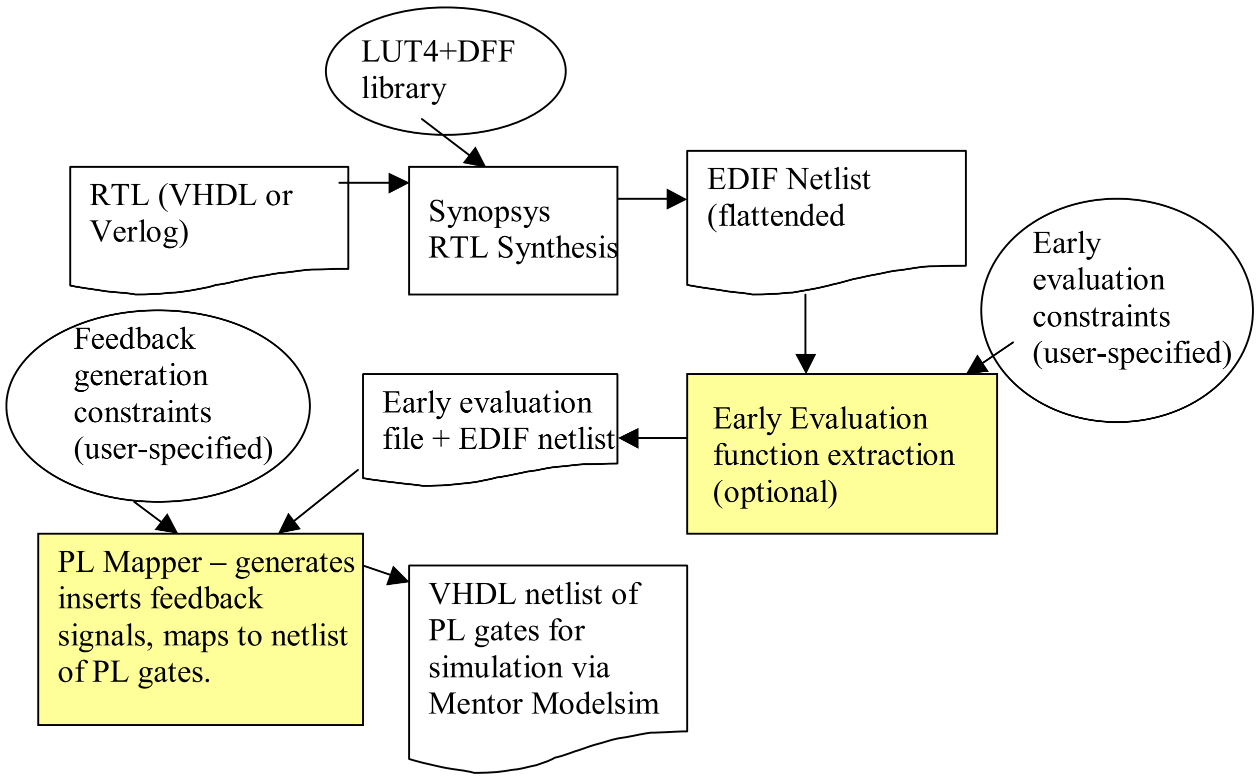

LUT4+DFF library
RTL (VHDL or Verlog)
Synopsys RTL Synthesis
EDIF Netlist (flattended
Early evaluation constraints (user-specified)
Feedback generation constraints (user-specified)
Early evaluation file + EDIF netlist
Early Evaluation function extraction (optional)
PL Mapper – generates inserts feedback signals, maps to netlist of PL gates.
VHDL netlist of PL gates for simulation via Mentor Modelsim

Figure 13. Tool Flow

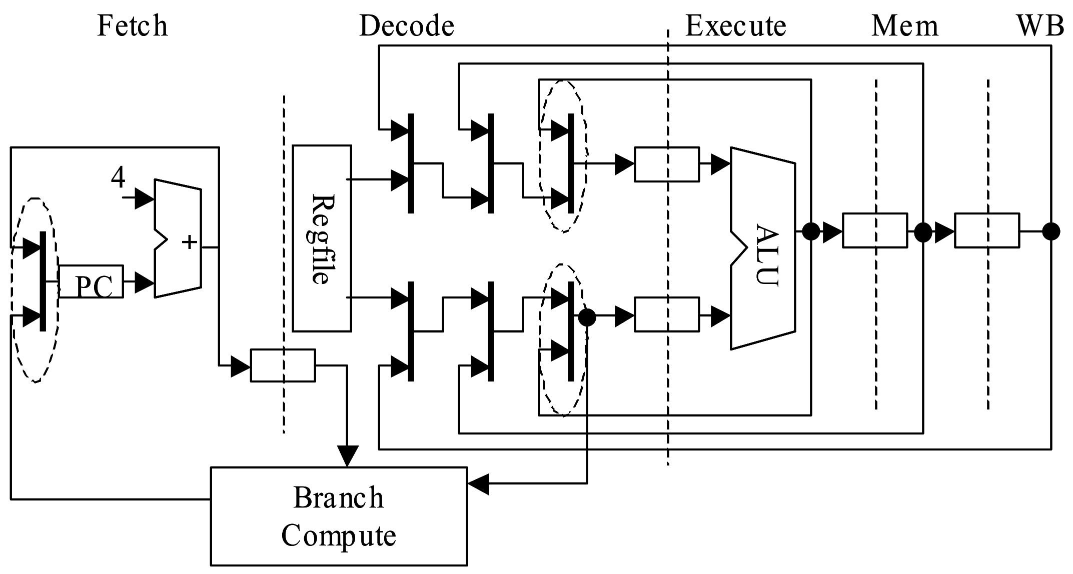


Figure 14. Pipelined CPU