Relative Timing Based Verification of Timed Circuits and Systems*

Hoshik Kim and Peter A. Beerel
EE-Systems Department
University of Southern California
Los Angeles, CA 90089
{hoshik, pabeerel}@eiger.usc.edu

Abstract

Advanced clock-delayed[1] and self-resetting domino circuits are becoming increasingly important design styles in aggressive synchronous as well as asynchronous design. Their design is particularly challenging due to the two-sided timing constraints needed to ensure their correctness. This paper proposes a relative-timing based technique to help verify such timed circuits. The main idea is to perform an untimed analysis to identify circuit paths for which delay-ordering constraints ensure that the circuit works. The timing verification of these delays is then reduced to standard simulation or a much simpler bounded delay analysis.

The benefit of this approach is three fold. First, it reduced the verification complexity exponentially by avoiding the modeling of explicit clocks. Second, it allows designers to design their circuits very aggressively, ensuring only that the relative path delays are met. Third, it facilitates fast incremental timing verification in response to design changes that do not affect the gate-level netlist.

1 Introduction

Many of aggressive timed (asynchronous) circuits and clock-delayed/self resetting domino synchronous circuits have demonstrated significant performance gains at the expense of complicated two-sided timing constraints that existing static timing analysis have difficulty in verifying [8, 9, 10, 11, 13, 14]. Designers have observed that the key property that must be met to ensure correct operations is the relative ordering of signal transitions [2, 3, 4]. In some cases, it is possible for the designer to identify these signal transitions and use them as constraints in a synthesis algorithm [3, 7]. In this paper, we focus on the associated verification problem. In particular, our verification goal is to find a (minimal) set of easily verifiable relative timing constraints necessary and sufficient to make the circuit work.

While recent researches have produced some good results on timing verification of these styles of circuits [5, 12], their methods still suffer from the limitations of explicitly modeling timing. The major problems, which are the motivation of our research, are the followings:

- They are limited to relatively small circuits because of the double exponential complexity of the general explicit-state based verification problem (state space + timing).
- Timing margins need to be modeled as min-max delays over all gates that are valid across process variations. This often forces designers to be more conservative than if they were required to meet constraints on relative path delays.
- Minor design changes that affect bounds require complete re-verification.

In the following section, we describe our new methodology and how it can overcome these limitations. Section 3 introduces the necessary background material to understand our algorithm to find the relative timing constraints. Section 4 shows some definitions used in our technique. Section 5 describes the algorithms in detail that generate a set of relative timing constraints. Section 6 illustrates the applications of our algorithms to a couple of examples and Section 7 concludes the paper.

2 Relative Timing (RT) based verification

2.1 Verification methodology

This section gives an intuitive overview of our relative timing verification methodology through the application to a simple circuit depicted in Figure 1.

Let us assume that a signal on node *B* control the output value of NOR gate (node *A*) and that there is a *positive* signal transition on *B*. This transition will trigger a *negative* signal transition on node *A* and a *positive* signal transition on node *x* consequently. If a *positive* signal transition on node *x* comes earlier than a *negative* signal transition on node *A*, a signal value on node *y* will be stable HIGH. If a *negative* signal transition on

[1] Also referred to as *Delayed-Reset Domino*

* This research has been partly supported by a grant from the Semiconductor Research Corporation #98-DJ-486, NSF grant CCR-9812164, and by a gift from the Intel Corporation.

node *A* comes earlier than a *positive* signal transition on node *x*, a signal value on node *y should* have a negative pulse. However, if a *positive* signal transition on node *x* comes later than a *negative* signal transition on node *A* but earlier than a *negative* signal transition on node *y*, then it is a hazard because a signal transition on node *y* is disabled without being fired after having been enabled. Consider the case where the stable HIGH and *negative* pulse are both acceptable circuit behaviors, but the runt pulse is not.

Explicit-timing based verification require the designer to provide upper and lower bounds of all gate delays and perform an expensive timing verification to deduce whether this circuit works. In contrast, we propose to find a set of relative timing constraints on path delays that guarantees the correctness of circuit operations. In particular, our algorithm yields the following type of constraints:

- If " — — " path delay is smaller than " ..... " path, *y* is stable high → OK;
- If " — — " path delay is larger than " ——— " path, *y* has negative pulse → OK;
- Otherwise, a runt pulse (or hazard) can occur → FAILURE.

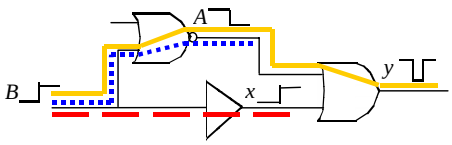

Figure 1: relative-timing constraints on path delays

It is important to note that these constraints are generated using an untimed reachability analysis of the circuit before layout [17, 18, 19]. Consequently, it avoids the more expensive timed reachability analysis. Nevertheless, the relative timing constraints still must be verified after layout. To do the latter, however, one can either use SPICE-level simulation or a simpler timing analysis of a reduced acyclic graph with bounded delays obtained via backannotation (e.g., [15]). Consequently, the hard part of the problem is to find the relative timing constraints, which is the focus of this paper.

More formally, the problem we propose to solve is the following. We assume the user describes a set of conditions in which the circuit should not exhibit. This description can either be implicit, such as the circuit should exhibit no hazards, or explicit, such as a set of states of the circuit that should not be entered. These conditions will be translated into a set of *failure transitions* in our behavioral model of the circuit. Then, our task is to find a set of relative timing constraints that have the following characteristics:

- All paths must have a common source. This feature makes the timing verification of these constraints intuitively easy to understand and perform.
- All paths must be minimal in length. This simplifies the analysis.
- The set of constraints must be necessary and sufficient to avoid the failure transitions. Sufficiency guarantees that all specified failure states will not be entered. Necessesity ensures that the relative-timing constraints do not prohibit the entrance of any other state, thereby maintaining maximal concurrency.

One problem is that such a set of relative timing constraints that satisfy the above features may not exist for some circuits and set of transitions. Consequently, in some cases we are forced to yield conservative constraints which may limit the concurrency of the circuit. For this reason, we also characterize a subset of circuits for which we believe we can obtain exact relative timing constraints.

2.2 Advantages of Relative Timing based verification

In general, the relative timing based approach has the following advantages over existing explicit-timing based approaches:

- **Reduces verification complexity exponentially**
  - Relative-timing based techniques do not need to model clocks. Thus, we can achieve the same complexity of the verification as that of untimed verification by avoiding explicitly modeling clocks.
  - This facilitates the use of mature symbolic BDD based methods, suggesting that we can handle significantly larger systems than possible using explicit-timing based approaches.
- **Facilitates tighter timing margins**
  - Relative timing constraints promote the use of circuits with tighter timing margins. Designers are likely to be more aggressive when verifying relative timing constraints than they are when verifying explicit upper and lower bounds of all gate delays [3]. One possible reason is that, in the former case, they can more easily consider the correlation of gate delays across process variation.
- **Promotes easy incremental verification**
  - Many minor design changes such as transistor sizing, layout, or technology/process migration can be easily verifiable (e.g., simulation or bounded delay analysis of acyclic graphs).

3 Background

Event-based models (such as Petri nets) explicitly represent the notions of concurrency and causality. On the other hand, state-based models (such as transition systems) provides a more natural formal framework to describe the behavior of sequential circuits and the behaviors that the circuit should not exhibit.

This work involves finding constraints on circuit paths that restrict the circuit behavior. Because the circuit paths are closely related to the causality in Petri nets we make use of both formalisms. In this section, we introduce the minimal background of transition systems (**TS**) and Petri nets (**PN**) used in this paper. For brevity, we assume that readers are familiar with transition systems and Petri nets, etc. For more detailed information, we refer the reader to [1, 6, 7].

3.1 Transition Systems (TS)

A *transition system* (**TS**) is a quadruple $TS = (S, E, T, s_{in})$, where S is a set of *states*, E is a set of *events*, T is a *transition relation*, $T \subseteq S \times E \times S$, and s_{in} is an *initial state*. A *transition* $(s, e, s') \in T$ can be denoted by $s \xrightarrow{e} s'$. Note that a state graph **SG**, which is often used to illustrate the specification and behavior of asynchronous circuits, is simply a binary encoded **TS**.

3.2 Petri nets (PN)

A *Petri net* (**PN**) is a quadruple $N = (P, T, F, m_0)$, where P is a set of *places*, T is a set of *transitions*, F is a *flow relation*, $F \subseteq (P \times T) \cup (T \times P)$, and m_0 is an *initial marking*. A marking is a function that assigns to every place a non-negative number of tokens. A transition $t \in T$ is enabled at a marking if all its input places have at least one token. When t is enabled, it may fire. The firing of t removes one token from each of its input places and deposits one token to each of its output places, leading to a new marking.

Note that Petri nets explicitly express the notions of concurrency, causality and conflict which are needed to derive relative timing constraints from behavioral models of sequential systems. Note also that exhaustive token flow analysis yields a transition system in which each state is a different marking.

4 Relative Timing Constraints

4.1 Failure transitions

In our methodology, the set of *failure transitions* is assumed to either be explicitly or implicitly specified by the user and is those set of transitions that the user does not want the circuit to execute. In the most common scenario, these transitions are implicitly specified to be the special **TS** failure transition which disables a signal transition on a node, indicating the possibility of a runt pulse (or hazard). In general, however, this set can be any subset of a **TS** transition relation *T*. In practice, these failure transitions will be identified via untimed reachability analysis of the circuit using existing formal verification techniques (e.g., [17, 18, 19]).

4.2 Delay of an event chain

An *event* of a **TS** is a signal transition in the specified circuit. Each arc of the corresponding state graph (**SG**) is labeled with an event, e.g., *B+*.

An *event chain* is a chain of events that occur along a *single path* of a circuit. For example, looking at Figure 1, we see that *B+A-y-* is an event chain.

The *delay of an event chain* is the sum of delays between consecutive events of an event chain, e.g., $D_{B+A-y-} = D_{B+A-} + D_{A-y-}$.

4.3 Event PNs and Time Separation of Events (TSE)

An *event **PN*** is an acyclic Petri net describing the causality of events. It contains a set of source transitions and a set of target transitions. All places are assumed to have fixed delays that model the corresponding gate delays. The *time separation of two events* (**TSE**) in an event **PN** is the delay between two events in an event **PN**. This delay can be expressed in terms of the delays of the places of the event **PN**. When the event **PN** has a single source place and no choice places, the **TSE** of an event pair is well defined and can be evaluated by assigning the source place to arrive at time zero and calculating the time that all events occur. When multiple source places exist, however, then only upper and lower bounds on the **TSE** can be found. To find expressions for these bounds we rely on the simple longest-path analysis described in [16]. Moreover, when the event **PN** contains choice places, each possible set of choices must be considered, also yielding bounds on the **TSE**.

Notice that in all cases the **TSE** expressions relate to delays of gates along circuit *paths* that connect the signals corresponding to the events.

4.5 Event Triples

In a transition system (**TS**) with a set of failure transitions (*FT*), an *event triple* (l, t, u) is defined as follows:

- t is a *target* event which is a label of a *failure* transition (causes a race),
  $t \in T = \{e \mid \exists s \xrightarrow{e} \in FT\}$;
  for *each* event t, let $Q = \{s \mid s \xrightarrow{t}\}$,
- l is a *lower* bound event which *enters* the set Q,
  $l \in L = \{e \mid \exists s \xrightarrow{e} s'$, where $s \notin Q$ and $s' \in Q\}$;
- u is an *upper* bound event which *escapes* from the set Q and is not in the set *FT*,
  $u \in U = \{e \mid \exists s \xrightarrow{e} s'$, where $s \in Q$, $s' \notin Q$ and $e \notin FT\}$.

The key idea of the paper is for a failure to occur the event t must occur after any event l that leads to Q and before each event u. Notice that if there are *multiple u* events for a pair of l and t events, the event t must occur before any of those u events for

there to be a failure. The intuition here is that if any of the *u* events occur before the *t* event, the circuit escapes from the dangerous set of states *Q*. Also, note that multiple *l* events for a given *t* should be treated separately because the circuit may enter *Q* through different *l* events. Notice how these conditions constrain the time at which *t* can occur from both sides, i.e., the constraint is two-sided.

These event-triple constraints can be formalized in terms of time separation of events conditioned on the set *Q*. In particular, for a given event triple, the conditions for a failure to occur is that $TSE(l, t) > 0$ and $TSE(t, u) > 0$, where the event *l* must enter *Q*. Our problem is thus translated to deriving expressions for $TSE(l, t)$ and $TSE(t, u)$ in terms of the delay of paths in the circuit. Notice also for relative timing constraints to exist for a given target failure event *t*, it must be the case that corresponding events *l* and *u* must exist. We will discuss conditions for this to be the case below.

4.6 Relative timing constraint for a target failure

In our methodology, multiple event triples may be formed for each target failure event. For each event triple, as will be explained below, we create an event **PN** with a minimal number of source, or *synchronization* transitions. By analyzing this event PN we derive expressions for two-sided timing constraints called a *relative timing* (RT) *constraint* that dictates when the target failure can occur.

The most important observation in this paper may be that two-sided constraints are needed to prevent only failure transitions, i.e., to avoid unnecessarily reducing concurrency thereby causing *false negatives*. In general, these relative timing expressions contain both minimum and maximum operations. In many circuits, however, they reduce to only two-sided constraints on the delay of an event chain, making them relatively easy to verify.

Consider the example in Figure 1. The target event PN to analyze is shown below:

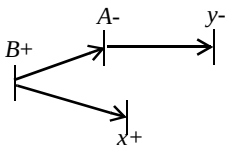

Figure 2: event PN for the circuit in Figure 1

The target event chain to verify the circuit is $B+x+$ because the correct circuit operation depends on the relative delay of the circuit path between *B* and *x* (with *B* and *x* rising) with respect to other path delays in the PN. Notice that since the PN has a single source it is easy to create a pair of simulation vectors to validate these constraints. Alternatively, a more sophisticated analysis of this acyclic graph using delay bounds [15] on gates or delay distributions can also be performed [16].

```
Generate_RT_constraints (TS, FT)
begin
    find_set_T;
    R = ∅; /* R is a set of RT constraints (failure conditions) */
    for each event t ∈ FT do
        find_set_L;
        find_set_U;
        if U is not empty then
            for each event l ∈ L do
                r = 1; /* r is a RT constraint */
                for each event u ∈ U do
                    /* r' is a sub-constraint */
                    r' = find_RT_constraint (l, t, u);
                    r = r ^ r';
                end for
                R = R ∪ {r};
            end for
        else print ("warning: states in Q will become unreachable");
            exit;
        end if
    end for
    return R;
end
```

Figure 3: The algorithm to generate RT constraints

5 Algorithms

This section describes preliminary algorithms we developed to support relative timing based verification. The algorithms try to address the following two core issues:

- **Existence:** Does a set of relative-timing constraints exist for which the circuit will work?**;**
- **Identification:** If so, find a (minimal) set of constraints that are sufficient (and necessary) to ensure the circuit works.

Figure 3 shows our algorithm to find a set of RT constraints (if one exists). The input of this algorithm is a **TS** and a set of failure transitions *FT*. It finds all target failures and forms an event triple for each failure. For each event triple, it calls the function *find_RT_constraint* (Figure 4) to find the corresponding RT constraint.

To generate an event **PN** in the routine *find_RT_constraint* we are currently exploring two possible approaches. The first

```
find_RT_constraint (l, t, u)
begin
find_event_PN (l, t, u); /* find an event PN from the event triple (l, t, u) */
find a (minimal) set of synchronization event SE from the event triple (l, t, u);
if SE contains more than one synchronization event se then
    print ("Warning: Constraints will be conservative");
end if
TSE_u (l, t) = expression for upper bound of TSE (l, t) [16];
TSE_u (t, u) = expression for upper bound of TSE (t, u) [16];
return the constraint {TSE_u (l, t) > 0} ^ {TSE_u (t, u) > 0};
end
```

Figure 4: The algorithm to find a RT constraint

approach relies on previous research showing that any elementary **TS** (**ETS**) can be mapped to a Petri net with a reachability graph isomorphic to the original **ETS** [1]. Fortunately, an **ETS** can be obtained from any non-elementary **TS** by *label splitting* [1]. Label splitting is the process of re-labelling events with the same label to remove explicit OR causality which cannot otherwise be easily expressed in Petri nets. Removing explicit OR causality also simplifies the task of finding our path constraints. One advantage of this work is that it uses symbolic techniques thereby allowing the analysis of very large state spaces. Unlike in [1], however, only part of the **PN** need be created. While this suggests that the computational cost of this step can be reduced, the details of the implementation of this step must still be explored. One potential problem with this approach, however, is that the assignment of delays to places is unclear when label splitting occurs. Moreover, this approach has high complexity since the process of forming the Petri net from an ETS involves solving an computationally expensive covering problem.

The second approach we are exploring is to adopt the work by Yoneda et al. [19,20] in which gates are modeled with a library of Petri nets. The main advantage of this approach is that creating the Petri net model of a circuit is straight forward. The circuit model, however, consists of a distributed set of Petri nets rather than a monolithic net. A second advantage of this approach is that the correspondence of delays on places and gate delays is pre-determined in the Petri net gate library. To create an event PN for a given event triple in this approach, we hope to use a form of backward Petri net unfolding on the distributed Petri net models. The details of this unfolding construction are part of our future work.

We hope to prove the following two properties of our algorithms:

- **Proposition 1** If a single synchronization point exists, the derived relative timing constraints removes only failure transitions;

- **Proposition 2** If the subset of states in the state graph **SG** reachable only through non-failure transitions is strongly connected, then event triples for each target failure will exist. If the events in each event triple have a common synchronization point then the constraints will be both necessary and sufficient.

The first proposition would guarantee that our identification procedure does not unnecessarily reduce the concurrency of the circuit. We believe that the key to the proof of this proposition, as mentioned earlier, will be our use of two-sided constraints.

If true, the second proposition would identify two conditions which must be satisfied for our algorithm to successfully find relative timing constraints. The first condition guarantees that for any target event *t* there will exist corresponding lower and upper bound events. The second condition is that all event triples should have a common synchronization point. Our future work includes a more careful refinement of these conditions.

6 Examples

6.1 Static C-element

Figure 5 shows an example of a static C-element [3]. Figure 5(a) shows a state graph (**SG**) specification. All failure transitions have been depicted to lead to special failure state labeled F for emphasis. In the formal state graph definition, the failure transitions actually lead to the appropriate binary labeled states. Figure 5(b) is a sum-of product C-element circuit implementation. Figure 5(c) is the **SG** behavioral model of the circuit which is the **TS** given to our algorithm. Figure 6(a) shows the event **PN**s for this example. Figure 6(b) illustrates the path delays of the constraints on the circuit found by our algorithm as follows.:

- Generate Chain Constraints:
  1. $T = \{B\text{-}, A\text{-}\}$
  2. For $t = B\text{-}$,
     $L = \{C+\}$, $U = \{u3+\}$
  3. Find an event PN and RT constraint for event triple
     $(C+, B\text{-}, u3+) \Rightarrow$
     $\{TSE\ (C+, B\text{-}) > 0\}$ and $\{TSE\ (B\text{-}, u3+) > 0)\}$
  4. Find event **PN** and express **TSE**s from common synchronization point: $C+$
     $TSE\ (C+, B\text{-}) = D_{C+B\text{-}} - D_{C+}$
     $TSE\ (B\text{-}, u3+) = D_{C+u3+} - D_{C+B\text{-}}$
  5. Thus, the constraint is: $D_{C+} < D_{C+B\text{-}} < D_{C+u3+}$
  Repeat Step 2 to 5 for $t = A\text{-}$
  6. $t = A\text{-}$, $L = \{C+\}$, $U = \{u2+\}$
  7. $(C+, A\text{-}, u2+) \Rightarrow$

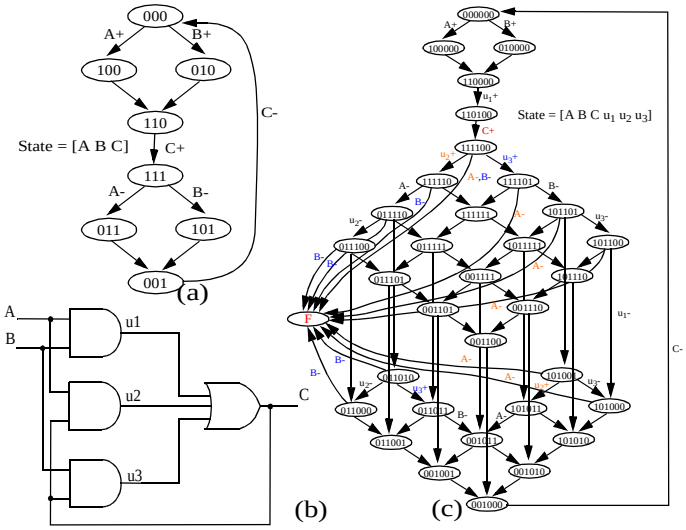

Figure 5: Static C-element: (a) specification **SG**, (b) sum-of-products C-element circuit, (c) behavioral model **SG**.

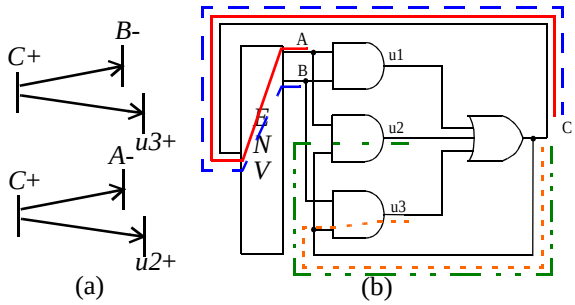

Figure 6: (a) Event **PN**s, (b) path delays on the circuit

{*TSE* (*C*+, *A*-) > 0} and {*TSE* (*A*-, *u2*+) > 0}

7. *TSE* (*C*+, *A*-) = $D_{C+A-} - D_{C+}$
   TSE (*A*-, *u2*+) = $D_{C+u2+} - D_{C+A-}$
9. Thus, the constraint is: $D_{C+} < D_{C+A-} < D_{C+u2+}$

Recall that the circuit will work "correctly" if and only if *none* of these RT constraints are satisfied

In this example, the lower bound of each RT constraint is trivially satisfied. This means that the derived RT constraint for this simple example is single-sided. An example of a circuit requiring two-sided constraints is shown next.

6.2 A circuit requiring two-sided constraints

Figure 7 shows an example of a circuit which allows an internal pulse but not a runt pulse, similar to the example in Figure 1. Figure 7(a) shows the **SG** specification, Figure 7(b) is a circuit implementation, and Figure 7(c) is a **SG** behavioral mode. Figure 8(a) shows the event **PN**s and Figure 8(b) illustrates the path delays corresponding to the derived RT constraints.

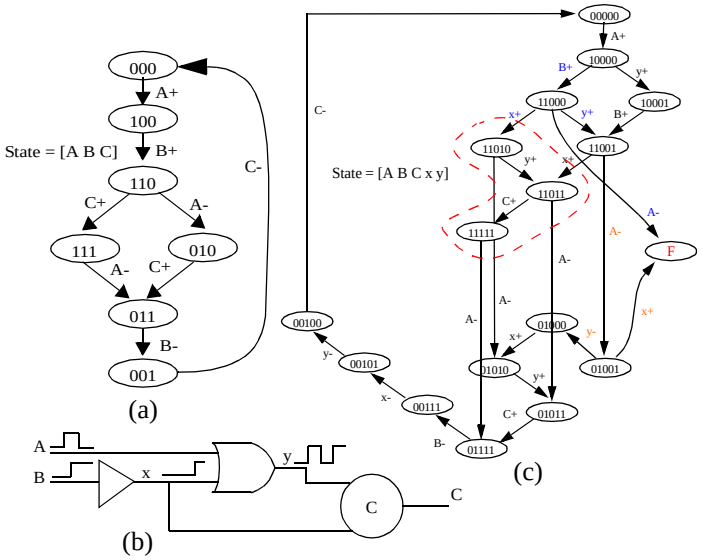

Figure 7: A circuit requiring two-sided constraints: (a) specification **SG**, (b) circuit, (c) behavioral model **SG**.

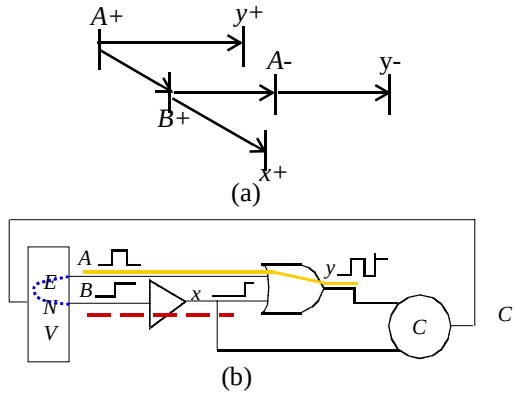

Figure 8: (a) Event **PN**, (b) path delays on the circuit

The following procedure describes the application of our algorithm:

- Generate Chain Constraints:

1. $T = \{A-, x+\}$
2. For $t = A-$,
   $L = \{B+\}$, $U = \{x+, y+\}$
3. For each event triple (l, t, u) form sub-constraints in terms of **TSE**s:
   (i) ($B+$, $A-$, $x+$) ⇒ *TSE* ($B+$, $A-$) > 0 and *TSE* (***A-***, $x+$) > 0
   (ii) ($B+$, $A-$, $y+$) ⇒ *TSE* ($B+$, $A-$) > 0 and *TSE* (***A-***, $y+$) > 0
4. Find event **PN** and expressions for **TSE**s
   (*i*) *TSE* ($B+$, $A-$) = $D_{B+A-} - D_{B+}$
   TSE (***A-***, $x+$) = $D_{B+x+} - D_{B+A-}$
   (ii) *TSE* ($B+$, $A-$) = $D_{A+B+A-} - D_{A+B+}$
   TSE (***A-***, $y+$) = $D_{A+y+} - D_{A+B+A-}$
5. Thus, the constraint is:
   $(D_{B+} < D_{B+A-} < D_{B+x+}) \wedge (D_{A+B+} < D_{A+B+A-} < D_{A+y+})$

Repeat Step 2 to 5 for $t = x+$:

6. For $t = x+$, $L = \{A-\}$, $U = \{y-\}$
7. ($A-$, $x+$, $y-$) ⇒ *TSE* ($A-$, $x+$) > 0 and *TSE* ($x+$, $y-$) > 0
8. *TSE* ($A-$, $x+$) = $D_{B+x+} - D_{B+A-}$
   TSE ($x+$, $y-$) = $D_{B+A-y-} - D_{B+x+}$
9. Thus, the constraint is: $D_{B+A-} < D_{B+x+} < D_{B+A-y-}$

This example shows two important features: (1) multiple sub-constraints in a single RT constraint when multiple *u* (escaping) events exist and (2) derivation of two-sided constraints (on D_{B+x+}) which are composed of three different path delays. Notice that if we had only a single-sided constraints here, such as $D_{B+x+} < D_{B+A-y-}$, we would remove good states, such as *states 11010*, *11011*, *11111* in this example, which in general may lead to worse circuit performance or false negatives.

For simplicity of exposition, these examples included only static gates. However, the proposed procedure can be directly applied to self-resetting and clock-delayed domino circuits by

creating corresponding gate models (e.g., a Petri net model of a dynamic NAND gate).

7 Conclusions

We presented novel verification techniques for supporting the design of aggressive timed and advanced domino circuits. These techniques identify a set of two-sided path delay constraints that are sufficient to guarantee the circuit works as expected.

We are currently refining and implementing these algorithms (using symbolic techniques) and hope to soon be able to demonstrate their effectiveness on both aggressively designed synchronous and asynchronous circuits and subsequently its utility to circuit designers.

Acknowledgments

The authors are grateful to Jordi Cortadella and Luciano Lavagno for insightful discussions on this work.

References

[1] J. Cortadella, M. Kishinevski, L. Lavagno, and A. Yakovlev. Synthesizing Petri Nets from State-Based Models. In *Proc. Int. Conf. on Computer-Aided Design*, Nov. 1995.

[2] R. Negulescu and A. Peeters. Verification of Speed-Dependences in Single-Rail Handshake Circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1998.

[3] K. Stevens, R. Ginosar, and S. Rotem. Relative Timing. To appear in *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1999.

[4] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev. RAPPID: An Asynchronous Instruction Length Decoder. To appear in *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1999.

[5] W. Belluomini, C.J. Myers, and H.P. Hofstee. Verification of Delayed-Reset Domino Circuits Using ATACS. To appear in *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1999.

[6] J. Cortadella, M. Kishinevski, A. Kondratyev, L. Lavagno, and A. Yakovlev. A Region-Based Theory for State Assignment in Speed-Independent Circuits. *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 793-812, Aug. 1997

[7] J. Cortadella, M. Kishinevski, A. Kondratyev, L. Lavagno, A. Taubin, and A. Yakovlev. Lazy transition systems: application to timing optimization of asynchronous circuits. In *Proc. Int. Conf. on Computer-Aided Design*, Nov. 1998.

[8] D. Van Campenhout, T. Mudge, and K. A. Sakallah. Timing Verification of Sequential Domino Circuits. In *Proc. Int. Conf. on Computer-Aided Design*, Nov. 1996.

[9] V. Narayanan, B. A. Chappell, and B. M. Fleischer. Static timing analysis for self-resetting circuits. In *Proc. Int. Conf. on Computer-Aided Design*, Nov. 1996.

[10] K. Bernstein, K. M. Carrig, C. M. Durham, P. R. Hansen, D. Hogenmiller, E. J. Nowak, and N. J. Rohrer. *High Speed CMOS Design Styles*. Norwell, MA: Kluwer Academic Publishers, 1998.

[11] G. Yee and C. Sechen. Clock-Delayed Domino for Adder and Combinational Logic Design. In *Proc. Int. Conf. on Computer Design*, Oct. 1996.

[12] W. Belluomini and C.J. Myers. Verification of timed systems using POSETs. In *Proc. Int. Conf. on Computer Aided Verification*. Springer-Verlag, 1998.

[13] K. Nowka, T. Galambos, and S. Dhong. Circuit design techniques for a Gigahertz integer microprocessor. In *Proc. Int. Conf. on Computer Design*, Oct. 1998.

[14] E. J. Shriver, D. H. Hall, N. Nassif, N. E. Raham, N. L. Rethman, G. Watt, and J. A. Farrel. Timing verification of the 21254: A 600 Mhz full-custom microprocessor. In *Proc. Int. Conf. on Computer Design*, Oct. 1998.

[15] S. Chakraborty and D. L. Dill. More Accurate Polynomial-Time Min-Max Timing Simulation. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1997.

[16] A. Xie, S. Kim and P. A. Beerel. Bounding Average Time Separations of Events in Stochastic Timed Petri Nets with Choice. To appear in *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, 1999.

[17] D. L. Dill. *Trace theory for automatic hierarchical verification of speed-independent circuits*. MIT Press, 1988.

[18] V. Vakilotojar and P. A. Beerel. Hiding Memory Elements in Induced Hierarchical Verification of Speed-Independent Circuits. In *Workshop Notes of International Workshop on Logic Synthesis*, 1998

[19] T. Yoneda and T. Yoshikawa. Using partial orders for trace theoretic verification of asynchronous circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1996.

[20] T. Yoneda and H. Ryu. Timed Trace Theoretic Verification Using Partial Order Reduction. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1999