Relative Timing Based Verification of Timed Circuits and Systems

Hoshik Kim* and Peter A. Beerel
Asynchronous CAD Group
University of Southern California
Los Angeles, CA, USA
{hoshik, pabeerel}@usc.edu

Ken Stevens
Strategic CAD Labs
Intel Corporation
Hillsboro, OR, USA
kstevens@ichips.intel.com

Abstract

Aggressive timed circuits, including synchronous and asynchronous self-resetting circuits, are particularly challenging to design and verify due to complicated timing constraints that must hold to ensure correct operation. Identifying a small, sufficient, and easily verifiable set of relative *timing constraints simplifies both design and verification. However, the manual identification of these constraints is a complex and error-prone process. This paper presents the first systematic algorithm to generate and optimize relative timing constraints sufficient to guarantee correctness. The algorithm has been implemented in our* RTCG *tool and has been applied to several real-life circuits. In all cases, the tool successfully generates a sufficient set of easily verifiable relative timing constraints. Moreover, the generated constraint sets are the same size or smaller than that of the hand-optimized constraints.*

1. Introduction

Aggressive timed synchronous and asynchronous circuit familes, including self-resetting domino circuits and GasP circuits, have demonstrated impressive performance gains [16, 21, 17, 18, 13] at the cost of requiring complicated two-sided timing constraints. These timing constraints must be considered during fanout optimization, transistor sizing, floorplanning, and routing. They must also be carefully verified pre- and post-layout. For these circuits to be widely adopted comprehensive CAD support is required.

One approach is to constrain the design of these circuits to simplify the problem. In particular, Sutherland et al. [22] suggested constraining the transistor sizing of GasP circuits to yield unit delays, thereby generating correct-by-construction transistor-level circuits and simplifying the pre-layout verification. This is an effective approach but overly constrains the circuits, reducing their potential benefit.

Other research has focused on the pre- and post-layout verification of these circuits. In particular, given a timed circuit with bounded-delays on all components, advanced verification tools can determine whether they will work correctly [3, 2]. These tools are limited to relatively small circuits because of the double exponential complexity of the exact timed state space based verification problem (state space + timing) [1]. Because timing margins must be characterized as min-max delays over all devices and environment delays that should be valid for all delay variations seen in today's deep-submicron processes [9], designers must be relatively conservative on how they set these delays. Moreover, even minor changes in transistor sizing, floorplanning, and/or routing that affect any of the delay bounds currently requires complete re-verification.

To overcome the above double exponential complexity problem of explicit timing, Peña et al. [14] proposed an approach performing an off-line timing analysis on a set of timed event structures that cover the failure traces rather than calculating the exact timed state space. In fact, as a side benefit of their verification approach the set of timing constraints used to prove the correctness of circuits can be reported for back-annotation. These constraints, however, are derived with given min-max delays on all gates. Consequently, this approach still suffers from overly conservative delay bounds and does not support incremental verification.

Recently, a relative timing based verification methodology was proposed to address these limitations. This methodology is based on the observation that the key property needed for correctness is often the relative ordering of signal transitions [11, 19, 16]. The idea is to decompose the timing verification problem into two parts. First, identify relative timing (RT) constraints sufficient to guarantee the correctness of a circuit using no min-max delays. Second, analyze the post-layout circuits to validate these timing constraints using a standard simulation technique or a

*This work was partly done during his internship at Strategic CAD Labs, Intel Corporation.

Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems (ASYNC'02)
1522-8681/02 $17.00 © 2002 IEEE

known polynomial-time bounded delay analysis technique given extracted min-max delays. Designers can also rely on the simpler simulation and/or analysis techniques to verify incremental changes to transistor sizing, floorplanning, and routing. Moreover, because the constraints are symbolic and thus not tied to min-max delays, it may also be feasible for the designers to verify the constraints using a technique that can take into account correlations among delays (e.g., random simulation), allowing them to design the circuits more aggressively.

The basic problem that this paper addresses is the generation of relative timing constraints. Currently the generation of relative timing constraints has been done manually relying on designers' intuition. While these manually generated constraints can be checked for sufficiency using an untimed verification tool [10, 20], the process is time-consuming and error-prone. This paper presents the first systematic algorithm to generate and optimize easily verifiable relative timing constraints sufficient to guarantee correctness. Both the generation and optimization of the constraints are based on an untimed analysis of the state space which uses state-of-the-art symbolic techniques to reduce run-time. The algorithm has been implemented in our RTCG tool and has been applied to several real-life circuits. In all cases, the tool successfully generates a sufficient set of easily verifiable timing constraints. Moreover, the generated constraint sets are the same size or smaller than that of the hand-optimized counterparts.

The organization of the rest of this paper is as follows. Section 2 describes the basic notions and formalism used throughout this paper. Section 3 and 4 describe the theory and algorithms for our relative timing constraint generator called RTCG in detail. Section 5 and 6 describe experimental results and conclusions.

2. Basic notions

This section introduces our formalism for modeling the behavior of circuits and their environment. We will also describe how we define the failure transitions that our relative timing constraints must avoid and several useful notions of reachability analysis.

We will use the GasP FIFO control circuit [21], illustrated in Figure 1, as a running example. Each PATH circuit controls the flow of data between pipeline stages implemented with single-rail logic. In particular, it latches single-rail data from its predecessor place to its successor place when its predecessor PLACE is FULL (encoded LO) and its successor PLACE is EMPTY (encoded HI). The latch signal is a pulse that must be wide enough to pass through data but not be too wide to allow the passing of two consecutive data tokens.

To analyze the behavior of the circuit, the behavior of the left and right environments must also be precisely defined. To do this, we have used signal transition graphs (STG) [5], as illustrated in Figure 2. To simplify the circuit, we have not modeled the keeper inverters on each PLACE because they do not affect the analysis of the circuit.

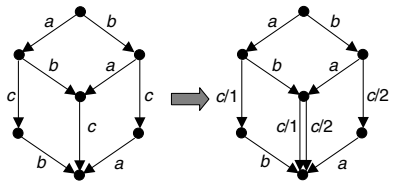

Figure 3. Label splitting

2.1. Transition systems

A *transition system* (TS) [12, 6] is a quadruple $TS = (S, E, T, s_{in})$, where S is a non-empty set of *states*, E is a non-empty set of *events*, T is a *transition relation*, $T = S \times E \times S$, and s_{in} is an *initial state*. We will denote the set of events corresponding to input and gate signals by E_I and E_G respectively. Therefore, $E = E_I \cup E_G$. A *state transition* $(s, e, s') \in T$ may be denoted by $s \xrightarrow{e} s'$.

A transition system provides a natural formal framework to describe the behavior of sequential circuits as well as behaviors that the circuit should not exhibit. Note that a state graph is simply a binary interpreted transition system where its transitions are labeled with an event in TS.

The transition system can be generated from the composition of the circuit and environment using standard untimed reachability analysis techniques [4, 7, 15, 24]. To avoid generating TS with OR causality, we use the well-known technique of label splitting to distinguish between different causes of the same event, that is, any event with multiple causes is separated into multiple events, each having a unique cause. For example, the event c in the TS shown in Figure 3 has two causes: a and b. By splitting this event into $c1$ and $c2$, each has its own unique cause. This splitting can easily be implemented during the generation of the TS and will simplify our subsequent analysis steps. Splitting, however, does make the interpretation of derived timing constraints on split events somewhat more challenging.

A simplified TS of the GasP FIFO control circuit and its environment shown in Figure 4. In particular, the TS is a projection of the entire TS onto a subset of the original signals (hiding signals *a*, *b* and *r*) to simplify the illustration. Interestingly, there was no OR causality in this circuit and thus label splitting was not necessary.

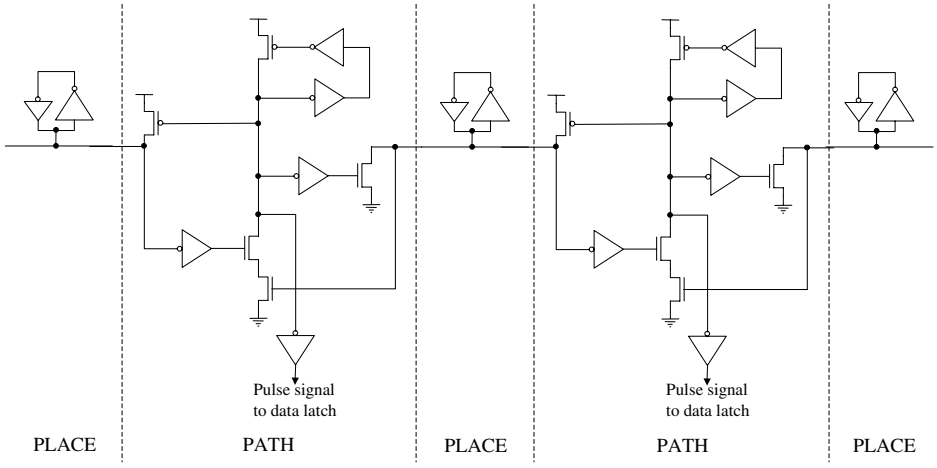

Figure 1. A GasP FIFO

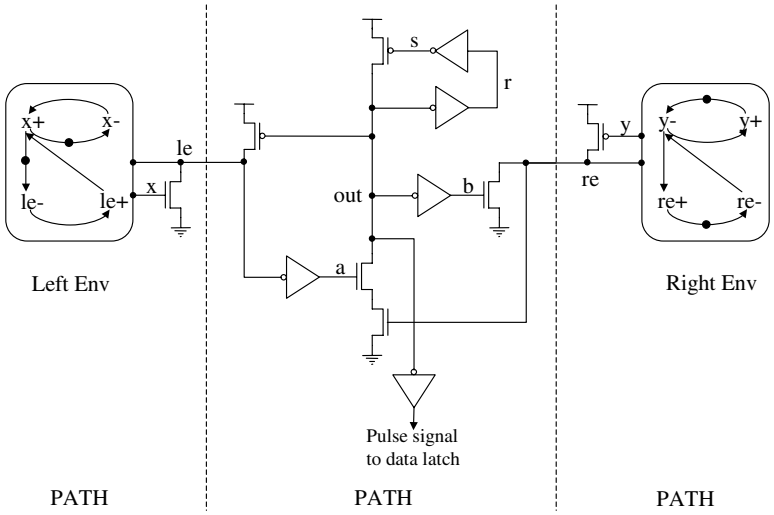

Figure 2. A GasP FIFO control circuit with the left and right environments modeled as STGs

2.2. Failure transitions

In our technique, *failure transitions* are assumed to be either explicitly or implicitly specified by designers. They are state transitions which designers do not want their circuits to execute. In the most common scenario, these transitions are implicitly specified to be state transitions leading to failure states, in which a circuit enables a signal transition that is not acceptable by the environment [15]. They are also often implied to be transitions which cause short-circuit currents in the circuit or transitions which violate the semi-modularity [23] of a signal in the circuit indicating the possibility of a runt pulse. In general, however, this set can be any subset of a transition relation, *T*, in a TS and denoted by T_{fail}. In practice, these failure transitions will be identified via untimed reachability analysis of the circuit using existing mature formal verification techniques (e.g., [4, 7, 15, 24]).

As an example, consider the GasP control circuit in Figure 2. After the first PATH modeled in the left environment uses the N-type transistor to drive the state conductor node *le* LO, the second PATH will take at least three gate-delays to drive *le* HI again using the P-type transistor. By the time the second PATH turns on the P-type transistor by triggering signal *out* LO to drive *le* HI, the first PATH will have turned off the N-type transistor. Otherwise, the circuit will be assumed to have a failure due to a short-circuit current on both transistors. In other words, if a signal transition *out-* fires earlier than *x-*, the circuit will have a failure. In the TS illustrated in Figure 4, this failure is modeled with

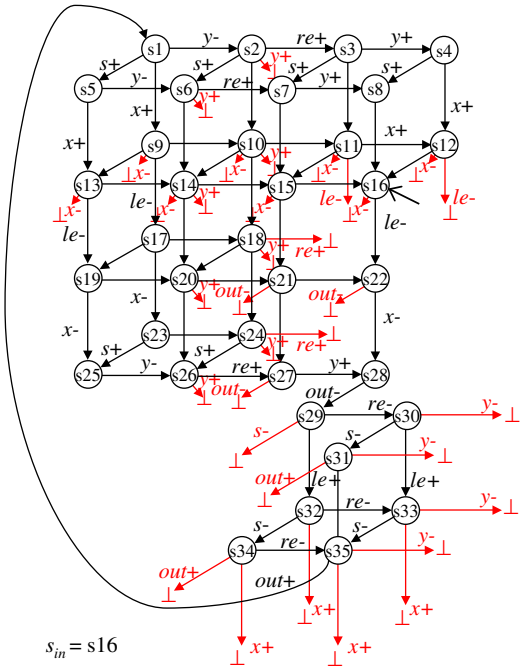

Figure 4. A simplified TS for the GasP control circuit and its environment.

two failure transitions labeled *out-* enabled at states $s21$ and $s22$ that lead to the symbol "$\perp$".

2.3. Reachability

To describe our methods we must introduce the following notations related to reachability analysis. A state s' is said to be *reachable* from a state s, denoted $s \rightsquigarrow s'$, if there is a (possibly empty) sequence of non-failure transitions starting in s and ending in s'. In addition, a state s' is said to be *reachable* from a state s within a set of states $r \subseteq S$, denoted $s \rightsquigarrow_r s'$, if there is a (possibly empty) sequence of non-failure transitions that remains within a given set of states r.

The forward reachability set of a set of states $S' \subseteq S$, denoted $\mathcal{F}(S')$, is the set of states reachable from S'. That is, $\mathcal{F}(S') = \{s \in S \mid \exists s' \in S' : s' \rightsquigarrow s\}$. Similarly, its backward reachability set $\mathcal{B}(S')$ of a set of states $S' \subseteq S$ is the set of states that can reach S'. That is, $\mathcal{B}(S') = \{s \in S \mid \exists s' \in S' : s \rightsquigarrow s'\}$. In addition, the forward reachability set $\mathcal{F}_r(S')$ of a set of states $S' \subseteq S$ constrained to a set of states r is the set of states reachable from S' within r. That is, $\mathcal{F}_r(S') = \{s \in S \mid \exists s' \in S' : s' \rightsquigarrow_r s\}$. Similarly, a constrained backward reachability set $\mathcal{B}_r(S')$ is defined as $\mathcal{B}(S') = \{s \in S \mid \exists s' \in S' : s \rightsquigarrow_r s'\}$.

It will also be useful to define the predicate $s \overset{\neg e}{\rightsquigarrow} s'$ to hold if s can reach s' without firing a transition labeled e. Correspondingly, we will let $\mathcal{B}_{\neg e}(S')$ denote the set of states that can reach S' without firing e for a set of states S' and event e. That is, $\mathcal{B}_{\neg e}(S') = \{s \in S \mid \exists s' \in S' : s \overset{\neg e}{\rightsquigarrow} s'\}$. Similarly, we define $\mathcal{B}_{r,\neg e} = \mathcal{B}_r \cap \mathcal{B}_{\neg e}$.

Lastly, it will be useful to define the one-step image computations. The forward image set $Img_e(S')$ of a set of states $S' \subseteq S$ for a given event $e \in E$ is the set of next states of S' by non-failure transitions labeled e. That is, $Img_e(S') = \{s \in S \mid \exists s' \in S' : (s', e, s) \in T - T_{fail}\}$. Similarly, The backward image set $Img_e^{-1}(S')$ of a set of states $S' \subseteq S$ for a given event $e \in E$ can be defined. That is, $Img_e^{-1}(S') = \{s \in S \mid \exists s' \in S' : (s, e, s') \in T - T_{fail}\}$.

3. Automatic generation of RT constraints

We divide the generation of relative timing (RT) constraints into two steps. In the first step, initial relative timing constraints are derived that guarantee no failure transition is reachable. These constraints effectively reduce only the necessary concurrency in the circuit to remove failure transitions. In the second step, the set of RT constraints is optimized through transformations that move and merge the constraints into a smaller or more easily verifiable set of RT constraints. This section describes the theory and algorithm for initial constraint generation. The subsequent section is devoted to constraint optimization.

The section starts with some definitions and formalism used in the following sections.

3.1. Qsets

A *Qset* of an event *e*, denoted Qset(*e*), is a set of states where a set of failure transitions labeled *e* are enabled to fire. A Qset is a set of dangerous states from which a failure transition may occur. Therefore, once the circuit enters the region of Qset(*e*) in a TS, the circuit must avoid such a failure transitions. Note that there can be different Qsets for one event.

We will define some notions and predicates here which are necessary to the formal definition of Qsets.

Definition 3.1 (Excitation region [8]) *A set of states S is called an excitation region of event e, denoted by ER(e), if it is a maximal (possibly disconnected) set of states in which for all $s \in S$ there is a transition $s \overset{e}{\rightarrow}$.*

For example, in the TS illustrated in Figure 4, ER($x+$)=$\{s1, s2, s3, s4, s5, s6, s7, s8, s32, s33, s34, s35\}$.

Definition 3.2 *Let TS = (S, E, T, s_{in}) be a TS. Let $S' \subseteq S$ be a set of states and $e \in E$ be an event. Let $T' = T - T_{fail}$ be a non-failure transition relation. The following predicates are defined for e and S':*

$$\mathbf{in}(e, S') \equiv \exists (s, e, s') \in T' : s, s' \in S'$$

$$\mathbf{enter}(e, S') \equiv \exists (s, e, s') \in T' : s \notin S' \wedge s' \in S'$$

$$\mathbf{exit}(e, S') \equiv \exists (s, e, s') \in T' : s \in S' \wedge s' \notin S'$$

$$\mathbf{false\_exit}(e, S') \equiv (\mathbf{in}(e, S') \vee \mathbf{enter}(e, S')) \wedge \mathbf{exit}(e, S')$$

Definition 3.3 (Qsets) *A set of states $q \subseteq \mathcal{F}(s_{in})$ in TS = (S, E, T, s_{in}) is called a Qset for event $e \in E$ and denoted by Qset(e) if $q \subset \mathsf{ER}(e) \wedge \forall e' \in E, \neg \mathbf{false\_exit}(e', q)$ holds. The event e is called the* target *event of the Qset.*

We restrict our Qset(e) to be only a proper subset of $\mathsf{ER}(e)$ because we want to have at least one transition labeled e that is potentially fireable in some reachable state. We restrict Qset(e) to have no false exit events to ensure that any exit event always exits the Qset. The motivation of this will be more clear when we discuss our specification of RT constraints in terms of event triples in Section 3.3.

Now let us consider a GasP circuit in Figure 2. The circuit will have a failure if the left environment tries to pull down the state conductor *le* by firing x+ before signal *out* has been reset, that is, *out*+. This failure is represented in the TS depicted in Figure 4 as the transitions labeled x+ from states $s32$, $s33$, $s34$ and $s35$. This set of states satisfies the definition of a Qset for event x+, denoted by Qset(x+) because it is a subset of $\mathsf{ER}(x+)$ and the only exit event, *out*+, is not a false exit.

3.2. Making a Qset

A key function in our algorithm involves finding the smallest Qset for a target event t that includes a given subset r of $\mathsf{ER}(t)$. The *make_Qset* function shown in Figure 5 performs this function by recursively adding to r only *essential states* that must be included in any Qset that includes r. In particular, as shown in Proposition 3.1, the forward image set $Img_{e'}(r)$ of r for a false exit event e' consists of essential states. The basic intuition behind this result is that the only way to remove the false exit events in a Qset that must include r is to add to r the sink states of any corresponding exit transition.

Proposition 3.1 (Essential states) *Let TS = (S, E, T, s_{in}) be a* TS *with a set of failure transitions $T_{fail} \subset T$. Let $r \subset \mathsf{ER}(e)$ for an event $e \in E$ and $r' \subset S$ be a Qset(e) such that $r \subset r'$. Let $e' \in E$ be an event such that $e' \neq e$. The following predicate holds:*

$$\mathbf{false\_exit}(e', r) \Rightarrow Img_{e'}(r) \subseteq r'$$

Proof Consider a state $s \in Img_{e'}(r)$ such that $(s', e', s) \in T - T_{fail}$ where $s' \in r$. $(\mathbf{in}(e', r) \vee \mathbf{enter}(e', r)) \wedge \mathbf{exit}(e', r)$ from $\mathbf{false\_exit}(e', r)$.

```
function make_Qset(e, r)
    /* e is the target event of a potential Qset */
    if (r ⊂ ER(e))
        if (∃e' ∈ E : false_exit(e', r));
            /* Recur adding essential states to r */
            return make_Qset(e, r ∪ Img_e'(r));
        else
            /* r is a Qset */
            return r;
        end if
    else
        /* No Qset can be found */
        return ∅;
    end if
end function
```

Figure 5. Algorithm for making a Qset from a given set of states

We have $\mathbf{in}(e', r') \vee \mathbf{enter}(e', r')$ from $\mathbf{in}(e', r) \vee \mathbf{enter}(e', r)$ because of $r \subset r'$ and the definitions of **in** and **enter**. Then, $\neg\mathbf{exit}(e', r')$ because of Definition 3.3 of Qsets. $s' \in r'$ because of $s' \in r$ and $r \subset r'$. Therefore, $s \in r'$ because of $\neg\mathbf{exit}(e', r')$. ■

3.3. Event triples

To formalize the definition of *event triples*, we first define a decomposition of a set of states into two parts, before and after the occurrence of some transition e. $\mathcal{S}_{\prec e} = \mathcal{B}_{S,\neg\overline{e}}(\mathsf{ER}(e))$ and $\mathcal{S}_{\succ e} = \mathcal{S} - \mathcal{S}_{\prec e}$. We then apply this to constrain the ordering of events by letting S be the excitation region of some event. That is, when we wish to constrain e to occur before t we must avoid firing any transition of t from the set of states $ER(t)_{\prec e}$. Notice that this set of states does not include states in $ER(t)$ that occur before $\overline{e}$. That is, the notion of *before* is reset by the presence of the $\overline{e}$.

An event triple is an ordered triple $\langle L, t, U \rangle$ associated with a Qset q for a target event t that causes a failure from some state in q. It consists of a set of *enter* events, the *target* failure event, and a set of *escape* events. An enter event $l \in L$ is an event associated with a non-failure transition entering q. An escape event $u \in U$ is an event that must satisfy two conditions: 1) u must be associated with non-failure transition that exits q and 2) $q_{\prec u}$ equals q. Intuitively, an escape event u is an event for which if constrained to occur before t, all failure transitions of t from q are avoided. This is formalized below.

Definition 3.4 (Event triple) *Let TS = (S, E, T, s_{in}) be a* TS *with a set of failure transitions $T_{fail} \subset T$. Let $L, U \subseteq E$ and $t \in E$. Let $q \subseteq \mathcal{F}(s_{in})$ be a Qset for event $t \in T_{fail}$,*

denoted Qset(t)*. The triple* $\langle L, t, U\rangle$ *is an event triple for* q *if all* $l \in L$ *and* $u \in U$ *satisfy:*

- $\texttt{enter}(l, q)$
- $\texttt{exit}(u, q) \ \wedge \ q_{\prec u} = q$
- *If* $t \in E_I$ *then* $u \in E_G$

The intuition of an event triple is that a failure will occur if t fires after any $l \in L$ and before all $u \in U$. As long as this condition is not satisfied, the failure associated with this Qset is avoided. In particular, we observe that the subset of states of $\mathsf{ER}(t)$ that occur after any $l \in L$ is $\bigcup_{l\in L} \mathsf{ER}(t)_{\succ l}$ and the subset of states of $\mathsf{ER}(t)$ that occur before all escape events is $\bigcap_{u\in U} \mathsf{ER}(t)_{\prec u}$. Our intuition implies that it is sufficient to remove only those transitions of t from states in the intersection of these two sets. However, this is not true if all $l \in L$ also occur after q in such a way that states in q from which we need to remove transitions of t are considered to occur before l. Consequently, for this case, we propose to expand the set of states from which we remove transitions of t to those that occur before all $u \in U$ and, in particular, do not check their relationship with $l \in L$.

The fact that a Qset must not have any false exit events ensures that the transition of an escape event leaves the Qset as desired. In addition, for Qsets with target events that are inputs, we find a non-input escape event. Otherwise, the event triple would imply a constraint that restricts the ordering of inputs signals and thus alter the circuit specification.

To prove failure-freedom, let $T_{L\prec t\prec U}$ denote the set of transitions we will associate with the event triple $\langle L, t, U\rangle$. Our goal is to define this set such that we can prove that if this set of transitions are avoided, the failure associated with the Qset is circumvented. Let T^C be the constrained transition relation derived from T by removing the transitions in $T_{L\prec t\prec U}$ associated with each event triple. Let $\mathcal{F}^C(s_{in})$ be the set of reachable states from s_{in} through the constrained transition relation T^C. Similarly, let $\mathsf{ER}^C(t)$ represent the constrained excitation region of t derived from $\mathsf{ER}(t)$ by removing states from which transitions of t have been removed from T. We then have the following theorem.

Theorem 3.1 *Let TS =* (S, E, T, s_{in}) *be a* TS *with a set of failure transitions* $T_{fail} \subset T$*. Given a Qset* $q \subseteq \mathcal{F}(s_{in})$ *with an event triple* $\langle L, t, U\rangle$*, let* $T_{L\prec t\prec U}$ *be defined as follows:*

1. *If* $\exists l \in L$ *such that* $\mathsf{ER}(t)_{\prec l} \cap q = \emptyset$*, then* $T_{L\prec t\prec U} = \{(s, t, s') \mid s \in \bigcup_{l\in L} \mathsf{ER}(t)_{\succ l} \cap \bigcap_{u\in U} \mathsf{ER}(t)_{\prec u}\}$.
2. *Otherwise,* $T_{L\prec t\prec U} = \{(s, t, s') \mid s \in \bigcap_{u\in U} \mathsf{ER}(t)_{\prec u}\}$.

Then, $\mathcal{F}^C(s_{in}) \cap ER^C(t) \cap q = \emptyset$*, i.e., failures associated with* q *are circumvented.*

Proof (By contraposition): Consider $s \in q \cap \mathcal{F}^C(s_{in})$. We know that $s \in q \Rightarrow s \in q_{\prec u}$ for any $u \in U$ because the definition of u implies that $q_{\prec u} = q$ This means that $s \in \mathsf{ER}(t)_{\prec u}$ because $q \subseteq \mathsf{ER}(t)$. Consequently, $s \in \bigcap_{u\in U} \mathsf{ER}(t)_{\prec u}$

Now consider the case for which $\exists l \in L$ such that $\mathsf{ER}(t)_{\prec l} \cap q = \emptyset$. This implies that $s \notin \mathsf{ER}(t)_{\prec l}$ because $s \in q$. It then follows that $s \in \mathsf{ER}(t) - \mathsf{ER}(t)_{\prec l}$ because $s \in q$ and $q \subseteq \mathsf{ER}(t) \Rightarrow s \in \mathsf{ER}(t)$. This means that $s \in \mathsf{ER}(t)_{\succ l}$ which implies that $s \in \bigcup_{l\in L} \mathsf{ER}(t)_{\succ l}$.

Consequently, we can conclude that the transition of t from s is in $T_{L\prec t\prec U}$ and thus $s \notin \mathsf{ER}^C(t)$. ■

Consider the Qset(x+) q for a target event x+ we discussed earlier for the GasP circuit in Figure 4. *le*+ is an enter event because there are three non-failure transitions labeled *le*+ which enter the Qset through $s32$, $s33$ and $s35$ respectively. *out*+ is an escape event because 1) there is a non-failure transition *out*+ which exits the Qset from $s35$ and 2) $q_{\prec out+}$ is $\{s32, s33, s34, s35\}$ which equals q. Thus, $\langle le+, x+, out+\rangle$[1] is an event triple for Qset(x+).

Let us now evaluate $T_{L\prec t\prec U}$. Note that $\mathsf{ER}(x+)_{\prec le+} = \mathcal{B}_{\mathsf{ER}(x+),\neg le-}(\mathsf{ER}(le+)) = \mathcal{B}_{\mathsf{ER}(x+),\neg le-}(\{s29, s30, s31\}) = \emptyset$, where $\mathsf{ER}(x+)$ = $\{s1, s2, s3, s4, s5, s6, s7, s8, s32, s33, s34, s35\}$. Thus, $\mathsf{ER}(x+)_{\succ le+} = \mathsf{ER}(x+) - \mathsf{ER}(x+)_{\prec le+} = \mathsf{ER}(x+)$. Similarly, it can be shown that $\mathsf{ER}(x+)_{\prec out+}$ = $\{s32, s33, s34, s35\}$. Consequently, $T_{L\prec t\prec U}$ = $\{(s, t, s') \mid s \in \mathsf{ER}(x+)_{\succ le+} \cap \mathsf{ER}(x+)_{\prec out+}\}$ = $\{s32, s33, s34, s35\}$ and all failure transitions of $x+$ from Qset(x+) will be removed. That is, the timing constraint represented by this event triple effectively prunes these failure transitions from the reachable state space. Consequently, if the Qset is entered, *out*+ must always be taken before the target event x+.

Note that the transition *out*+ enabled at state $s34$ in the Qset(x+) is another failure transition. This failure transition will be effectively pruned by the event triple of the Qset(*out*+) that consists of state $s34$.

One potential issue with the semantics of an event triple is that while the constrained ordering of events is necessary to avoid a failure only in the Qset for which it is defined, the constraint applies to the entire state space. Consequently, the constraint may unnecessarily make some states unreachable, reducing concurrency, in many other parts of the state space. In practice, however, we have not found this unnecessary reduction in concurrency to be a significant problem.

3.4. Finding Initial Qsets and Event Triples

Our algorithm for finding the initial Qsets and corresponding event triples is as follows. First, for each prop-

[1] For simplicity, we omit the set notation for singleton L's and U's.

erty of interest we identify the events involved in the corresponding failure transitions. For each such event, we use the *make_Qset* function to create a Qset that includes the source state of the corresponding failure transitions. Note that multiple Qsets of the same event may be created if that event violates multiple properties. The decomposition of Qsets based on property violations, however, increases the chance of finding reasonable escape events. Note that if *make_Qset* fails to find a Qset, the algorithm aborts indicating that no relative timing constraints can be found that are sufficient for correctness.

For any Qset q with no escape event, we create a new Qsets for each enter event e of q. In particular, we use *make_Qset* to create a Qset for e that includes the source of all the entering state transitions. The basic idea is that the constraints associated with the event triples of the new Qsets will implicitly ensure that the failure transitions from q cannot occur. We say the new Qsets *cover* q. We repeat this procedure until all Qsets have either at least one escape event or no enter events. If, however, during this procedure, a Qset with no escape event is encountered that contains s_{in}, the algorithm aborts indicating that no relative timing constraints could be found.

We further examine Qsets obtained that have no enter event. There are three cases to consider. The first case is where we have a Qset with no enter event that does not include the initial state. The only possibility for this Qset is that the transitions that enter this set are newly defined failure transitions. In this case, this Qset can only be reached by failure transitions which implies that it is covered by other Qsets and can be discarded from consideration. The second case is where the Qset has the initial state s_{in} and has an escape event. In this case, the Qset is assigned a special *reset* enter event that represents the initial power up of the circuit. The last case is where the Qset contains s_{in} but has no escape event. In this case, upon reset, there is no constraint that can avoid the failure, and the procedure aborts indicating that no relative timing constraints could be found.

If the above procedure is successful, each remaining Qset has one target event and possibly many escape and enter events. Consequently, it follows from Theorem 3.1 that the set of relative timing constraints implied by the identified event triples guarantees the circuit avoids all failures. When the specification is mirrored, forming a closed system, this also implies that the circuit conforms [7, 15] to the specification. The number of initial constraints, however, may be very large, motivating the constraint optimizations discussed in the next section.

4. Optimization of relative timing constraints

The relative timing paradigm [19] allows the existence of many possible sets of timing constraints that are sufficient to guarantee the correctness of circuits. Each relative timing constraint effectively reduces the concurrency allowed by the circuit by constraining the relative ordering of otherwise concurrent signal transitions. The initial constraint generation method described above finds a set of relative timing constraints whose goal is to minimally reduce concurrency while still ensuring correctness. The result, however, is a set of constraints which may be very large and difficult to validate. This section discusses methods to optimize the set of initial constraints into a smaller set of typically stronger constraints which is more easily verifiable. These methods can be used to tradeoff additional reduction in concurrency for reduced complexity of constraints.

In particular, we present two basic procedures for strengthening a RT constraint by manipulating Qsets. The first involves creating new failure transitions and a new Qset that covers an existing Qset in a procedure we call *move_Qset*. The second involves merging existing Qsets for the same target event, yielding a typically smaller set of event triples for the merged Qset, in a procedure we call *merge_Qsets*.

4.1. Moving a Qset

Consider a constraint implied by an event triple $\langle x+, le-, s+\rangle$ (implying that *le-* should not fire before $s+$ and after $x+$) for the Qset(*le-*) $q = \{s11, s12\}$ in Figure 4. This timing constraint can be *strengthened* by constraining the preceding transition $s+$ enabled at $s1$, $s2$, $s3$ or $s4$ to occur before $x+$. In general, the process of strengthening a constraint for a target event t_2 involves creating a Qset that contains the sources of all state transitions of some event t_1 that *precede* transitions of t_2 and avoiding t_2 by creating constraints that avoid t_1. Note that timing constraints associated with input target events should not be strengthened since this may result in improperly altering the circuit specification.

Function *move_Qset*, shown in Figure 6, implements this strengthening process. Let us consider the Qset(*le-*) $q = \{s11, s12\}$ again and try to move q to a new Qset for a target event $x+$. The function first verifies that q's target event *le-* is not an input event. It then determines that q is unreachable from the initial state $s16$ without firing a transition $x+$ because the backward reachability set of q without firing $x+$, $\mathcal{B}_{\neg x+}(q) = \{s11, s12\}$, does not include $s16$. Thus, it calls *make_Qset* to try to make a new Qset q' for the target event $x+$ from the initial set of states $Img^{-1}_{x+}(\mathcal{B}_{\neg x+}(q)) = \{s1, s2, s3, s4\}$. Because $Img^{-1}_{x+}(\mathcal{B}_{\neg x+}(q))$ already satisfies the Qset condition in Definition 3.3, *make_Qset* returns $q' = \{s1, s2, s3, s4\}$. Since there is an escape event $s+$ from q', *move_Qset* returns the Qset$(x+)$ q' as a new Qset which covers q. The constraint implied by the event triple $\langle out+, x+, s+\rangle$ from the new Qset q' means that $x+$ should

```
function move_Qset(e, q)
    t := the target event of q;
    /* Do not move a Qset for an input target event */
    if (t ∉ E_I)
        if (s_in ∉ B_¬e(q))
            q' := make_Qset(e, Img_e^-1(B_¬e(q)));
            if there is an escape event from q'
                return q';
            end if
        end if
    end if
    return ∅;
end function
```

Figure 6. Algorithm for moving a Qset to a new target event

not fire before $s+$ and after $out+$. As expected, this constraint avoids the failure transitions.

4.2. Merging a Qset

Consider a constraint implied by an event triple $\langle out+, x+, s+\rangle$ for the Qset($x+$), $\{s1, s2, s3, s4\}$, relabeled q_1 for convenience, that is newly created by moving q in the previous section and a constraint implied by an event triple $\langle le+, x+, out+\rangle$ for the Qset($x+$) $q_2 = \{s32, s33, s34, s35\}$ in Figure 4. These two constraints can be satisfied if a transition $s+$ enabled at states in q_1 is constrained to occur before $x+$ enabled at states in $q_1 \cup q_2$ after firing $le+$. This effectively merges the two constraints implied by the neighboring event triples $\langle out+, x+, s+\rangle$ and $\langle le+, x+, out+\rangle$.

Function *merge_Qsets*, shown in Figure 7, implements this merging process. Observe the Qsets $q_1 = \{s1, s2, s3, s4\}$ and $q_2 = \{s32, s33, s34, s35\}$ again in Figure 4. Because they have the same target event $x+$, it calls *make_Qset* to try to create a new Qset q' for event $x+$ from $q_1 \cup q_2$. Because $q_1 \cup q_2$ already satisfies the Qset condition in Definition 3.3, *make_Qset* successfully returns $q' = \{s1, s2, s3, s4, s32, s33, s34, s35\}$. Since there is an escape event $s+$ from q', *merge_Qsets* returns Qset($x+$) q' as a new Qset which covers both q_1 and q_2. Now the event triple for q', $(le+, x+, s+)$, implies a new RT constraint. The failure is avoided if $x+$ does not fire before $s+$ and after $le+$. However, because $le+$ is the causal predecessor of $x+$, $le+$ will always fire before $x+$. Thus, the constraint effectively only imposes the ordering that $s+$ should fire before $x+$. As expected, this constraint makes the constraints associated with q_1 and q_2 redundant. Note that strengthening a constraint may also be able to make other unrelated constraints redundant.

```
function merge_Qsets(q_1, q_2)
    e_1 := the target event of q_1;
    e_2 := the target event of q_2;
    if (e_1 = e_2)
        q' := make_Qset(e_1, q_1 ∪ q_2);
        if there is an escape event from q'
            return q';
        end if
    end if
    return ∅;
end function
```

Figure 7. Algorithm for merging Qsets for the same target event

4.3. Top level algorithm

The optimization procedures *move_Qset* and *merge_Qsets* are used both to reduce the overall number of constraints and to support hierarchical verification by moving Qsets to primary input target events. For example, a constraint that refers to an internal signal, which may be difficult to observe and control, can be converted to one that refers to primary inputs to simplify the validation of the constraints and support hierarchical timing analysis. In particular, such constraints may make it possible to constrain only the environment of the circuit and not require the circuit to be re-designed. In the future, the moving of Qsets can be guided by rough timing analysis (e.g, using timed event structures such as in [14]) that could identify which constraints can be strengthened without unduly constraining the relative delays of circuit components. Currently, our application of these optimizations is rather straightforward. We try to move all Qsets to primary inputs and then merge Qsets with the same target events, removing any constraint that becomes redundant.

5. Experimental results

We have implemented the algorithms described in the previous sections into our tool `RTCG` using symbolic BDD-based techniques for reachability analysis. All experiments have been performed on a 450Mhz 1GB Ultra SPARC60 machine. The properties verified for each circuit consist of semi-modularity, conformance to its specification, and the absence of prolonged short-circuit current. In all cases, we have formally verified that the derived relative timing constraints guarantee failure-freedom by performing reachability analysis on the constrained transition system.

We applied our algorithm to the asynchronous benchmark circuits generated using standard synchronous technology mapping of speed-independent complex-gate cir-

cuits [14]. Consequently, these examples are not hazard-free under the unbounded delay model. We also tested several real-life industrial circuit blocks [16, 21, 17] for which our generated RT constraints have been further verified using Analyze [20]. Moreover, the generated constraint sets for these circuits are the same size or smaller than that of the hand-optimized constraints. For all circuits tested, the tool successfully generated a sufficient set of relative timing constraints.

Table 1 shows the obtained results. The second and third columns lists the number of signals and gates in the circuit. The fourth column reports the number of untimed reachable states. The fifth and sixth columns report the numbers of initial and optimized Qsets. Finally, the last column labeled cpu shows run times given in seconds.

5.1. A GasP FIFO control circuit

This section discusses the RTCG generated RT constraints for the GasP FIFO control circuit in Figure 2. Note that although our running example used a simplified TS, the tool generated these results from the complete TS. The results are given in Table 2 which shows a list of Qsets and corresponding event triples generated for this circuit. The second column indicates the size of the Qset in terms of number of states and the last column lists the associated event triples.

Consider the constraint implied by the event triple $\langle x+, x-, le- \rangle$ for Qset q_1. It captures the timing assumption we need to have for the left environment, stating that once x turns on the N-type transistor to drive le LO, x should not go LO before le goes LO. In addition, consider the event triple $\langle re-, y-, b- \rangle$ for Qset q_2. This constraint avoids the short-circuit current on the state conductor re, stating that once re goes LO, by the time the right environment turns on the P-type transistor by firing $y-$, the N-type transistor driving re should have been turned off by firing $b-$.

6. Conclusions

This paper presented a novel relative-timing technique to facilitate the design and verification of emerging high performance circuit design methodologies that rely on complex two-sided timing constraints. In particular, this paper presents the first automatic procedure for generating and optimizing a set of RT constraints sufficient to ensure correctness from the untimed specification of the circuit. The procedure has been implemented in our tool RTCG and has been applied to several real-life circuits. For all circuits tested, the tool successfully generates a sufficient set of relative timing constraints. Moreover, the generated constraint sets are the same size or smaller than that of the hand-optimized constraints. Our future work includes the characterization of the class of circuits for which the relative timing constraints are appropriate and the exploration of techniques to guide practical design choices that satisfy these constraints.

Qset ID	size in # states	event triples		
		enter(s)	*target*	*escape(s)*
q_1	29	$x+$	$x-$	$le-$
q_2	33	$re-$	$y-$	$b-$
q_3	21	$le+, out-$	$x+$	$out+$
q_4	24	$y-$	$y+$	$re+$
q_5	8	$a+, y-$	$re+$	$s+$
q_6	5	$a+, re+, x+$	$out-$	$x-$
q_7	3	$s-$	$out+$	$re-$
q_8	8	$r+$	$s-$	$a-$
q_9	12	$le-, re+$	$a+$	$s+$
q_{10}	7	$out-$	$b+$	$y+$

Table 2. RT constraints for the GasP circuit

Acknowledgement

The authors would like to thank Jordi Cortadella for providing his asynchronous benchmarks for invaluable comments on this work. We also thank the anonymous reviewers of earlier versions of this work who have motivated numerous theoretical and practical improvements. This research has been partly supported by NSF grants CCR-9812164 and 53-4503-0640 and gifts from TRW and Fulcrum Microsystems.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] W. Belluomini. *Algorithms for Synthesis and Verification of Timed Circuits and Systems*. PhD thesis, Department of Computer Science, University of Utah, Sept. 1999.

[3] W. Belluomini, C. J. Myers, and H. P. Hofstee. Verification of delayed-reset domino circuits using ATACS. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 3–12, Apr. 1999.

[4] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4):401–424, April 1994.

[5] T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*. PhD thesis, MIT Laboratory for Computer Science, June 1987.

[6] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri nets from state-based models. Technical report, Universitat Politècnica de Catalunya, 1995.

name	# signals	# gates	# untimed states	# init. Qsets	# opt. Qsets	cpu
GasP-control	9	7	148	17	10	2.74
global-STP-Natomic	6	5	77	14	10	1.42
global-STP-Nglobal	5	3	380	11	7	1.23
RAPPID-ByteCtl	7	4	155	16	14	3.25
clk-gen	10	9	128	8	3	0.84
rcv-setup	11	8	105	12	8	4.17
sbuf-read-ctl	18	15	131	9	4	6.76
alloc-outbound	20	16	139	9	3	13.08
mp-forward-pkt	13	10	125	3	3	0.89
chu133	12	9	198	15	3	4.33
dff	8	6	173	32	28	27.17
sbuf-send-ctl	16	13	633	22	16	41.39
sbuf-send-pkt2	17	13	481	25	16	69.97
converta	14	12	559	25	24	113.12
ram-read-sbuf	23	18	2428	25	13	127.98

Table 1. Experimental results

[7] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

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

[9] C. J. Myers, T. G. Rokicki, and T. H.-Y. Meng. Automatic synthesis and verification of gate-level timed circuits. Technical Report CSL-TR-94-652, Stanford University, Jan. 1995.

[10] R. Negulescu. *Process Spaces and Formal Verification of Asynchronous Circuits*. PhD thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Aug. 1998.

[11] 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*, pages 159–170, 1998.

[12] M. Nielsen, G. Rozenberg, and P. S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3–33, 1992.

[13] K. J. Nowka and T. Galambos. Circuit design techniques for a gigahertz integer microprocessor. In *Proc. International Conf. Computer Design (ICCD)*, pages 11–16, Oct. 1998.

[14] M. A. Peña, J. Cortadella, A. Kondratyev, and E. Pastor. Formal verification of safety properties in timed circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 2–11. IEEE Computer Society Press, Apr. 2000.

[15] O. Roig. *Formal Verification and Testing of Asynchronous Circuits*. PhD thesis, Univsitat Politècnia de Catalunya, May 1997.

[16] 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. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 60–70, Apr. 1999.

[17] D. Sager, G. Hinton, M. Upton, T. Chappel, T. D. Fletcher, S. Samaan, and R. Murray. A 0.18um CMOS IA32 microprocessor with a 4GHz integer execution unit. In *International Solid State Circuits Conference*, pages 324–325, Feb. 2001.

[18] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins. Asynchronous Interlocked Pipelined CMOS circuits operating at 3.3-4.5GHz. In *International Solid State Circuits Conference*, pages 292–293, Feb. 2000.

[19] K. Stevens, R. Ginosar, and S. Rotem. Relative timing. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 208–218, Apr. 1999.

[20] K. S. Stevens. *Practical Verification and Synthesis of Low Latency Asynchronous Systems*. PhD thesis, Dept. of Computer Science, University of Calgary, Canada, Sept. 1994.

[21] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 46–53. IEEE Computer Society Press, Mar. 2001.

[22] I. Sutherland and J. Lexau. Designing fast asynchronous circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 184–193. IEEE Computer Society Press, Mar. 2001.

[23] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli. A unified signal transition graph model for asynchronous control circuit synthesis. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 104–111. IEEE Computer Society Press, Nov. 1992.

[24] 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*. IEEE Computer Society Press, Mar. 1996.