A Path-Oriented Approach for Reducing Hazards in Asynchronous Designs

Meng-Lin Yu and P. A. Subrahmanyam
AT&T Bell Laboratories, Holmdel, N.J. 07733

Abstract

We describe a path-oriented approach for analyzing hazards in an asynchronous circuit that is synthesized from a specification in the form of a signal transition graph. We also describe a technique to avoid hazards by adjusting the delays in a circuit; rather than doing this piecewise for each path, we show how improved results can be obtained by solving an appropriately formulated linear programming problem.

1 Introduction and Motivation

A hazard in an asynchronous circuit implementation refers to an unexpected “glitch” in a signal value, i.e., a transition that is not allowed by the circuit specification. Hazards can lead to malfunctions in a system by triggering an erroneous sequence of transitions (possibly leading to erroneous states). It is therefore important to avoid hazards in an asynchronous circuit implementation. In so far as is feasible, it is desirable to detect the potential for hazards at the level of a circuit “specification” rather than analyzing the synthesized circuit implementation.

A convenient way to specify an asynchronous circuit behavior is via a Signal Transition Graph (STG) in which vertices represent signal transitions and arcs represent causal dependencies between signal transitions[1]. Synthesis is then performed by mapping the STG into a state graph, and eventually into an asynchronous circuit.

In this paper, we describe a path-oriented approach for analyzing hazards in an asynchronous circuit that is synthesized from an STG specification. For many designs, such an approach provides a good visual way of determining potential hazards, and assists designers in assessing a design and making appropriate changes. We also describe a technique for avoiding hazards by adjusting the delays in a circuit; rather than doing this piecewise for each path, we show how improved results can be obtained by solving an appropriately formulated linear programming problem. In addition to the benefits of working at a higher level of abstraction, this method handles some classes of dynamic hazards in addition to static hazards.

The paper is organized as follows: Section 2 summarizes some of the relevant concepts and definitions from literature. Section 3 discusses a path-oriented approach for analyzing hazards. Section 4 describes how this technique can be used to avoid hazards by formulating and solving an appropriate linear programming problem. Section 5 contains preliminary experimental results, while Section 6 draws some conclusions.

2 Preliminaries and Background

This section summarizes some of the relevant concepts relating to Petri nets and signal transition graphs.

Petri Nets. A Petri net is a triple consisting of a finite set of transitions \( T \), a finite set of places \( P \) and a binary relation between transitions and places called a flow relation \( F \subseteq T \times P \cup P \times T \).

A Petri net thus defines a bipartite graph, and is commonly pictorially depicted as such. The nodes are either places or transitions, and arcs between the nodes in \( P \) and \( T \) represent the relation \( F \). If \((p, t) \in F\), an arc is drawn from place \( p \) to transition \( t \). If \((t, p) \in F\), an arc is drawn from transition \( t \) to place \( p \). If \((v_1, v_2) \in F\) then \( v_1 \) is called a predecessor of \( v_2 \), and \( v_2 \) is called a successor of \( v_1 \).

A free-choice net (FC net) is a Petri net, where, if a place \( p \) has more than one successor, say \( t_1, \ldots, t_n \), then \( p \) is restricted to be the only predecessor of \( t_1, \ldots, t_n \).

Signal Transition Graphs. A signal transition graph is an interpreted free-choice Petri Net. Transitions of the petri net are interpreted as signal transitions, i.e., changes of values on input, output or internal signals. Places are interpreted as conditions on the signal states.

A positive or rising transition of a signal \( s \) is denoted \( s^+ \); a negative or falling transition is denoted \( s^- \). Henceforth, we use \( s^* \) to denote any transition (either \( s^+ \) or \( s^- \)) of the signal \( s \). In such a case, \( s^* \) denotes the transition complementary to \( s^* \), i.e., either \( s^- \) or \( s^+ \).

Dynamic Behavior of a Petri Net/STG: Marking and Firing. A token marking of a petri net is an assignment of a non-negative integer to each place in the net. Tokens are typically depicted by means of dots in the place, or along the edges leading to a place/transition.

A transition is enabled, implying that the corresponding event can actually occur in the system, when there is at least one token on each place (equivalently, the fanin arc) leading into the transition.

Once a transition is enabled, it must eventually fire. When a transition fires, one token is removed from each predecessor of the transition, and one token is placed in each successor of the transition.

A set of transitions is said to be concurrent if all the transitions in the set are simultaneously enabled in some marking reachable from the initial marking. If \( t_1 \) is constrained to occur after \( t_2 \), we write \( t_1 \Rightarrow t_2 \); \( t_1 \) and \( t_2 \) are said to be (sequentially) ordered. Transitions that are neither concurrent nor sequential are said to be in conflict.

Live and Safe Nets. A free-choice net is said to be live if it is always the case that each of its transitions is enabled in some marking that is reachable from its initial marking, i.e., no transition is permanently disabled in a live net. A free-choice net is said to be safe if no place is

29th ACM/IEEE Design Automation Conference®
ever assigned more than one token after any sequence of firings from the initial marking. All of the STGs considered in this paper will be assumed to be live and safe.

State Graphs. A state graph is an alternative representation of the behavior of an asynchronous circuit, where all of the concurrency in the transitions is resolved by representing it as a set of interleaved sequential transitions. The state graph corresponding to an STG can be computed by starting with the initial marking of the STG, and exhaustively simulating the dynamic behavior of the STG. Figure 1 shows an example STG and its state graph.

Unique State Coding. The state graph SG corresponding to an STG is used in the synthesis technique described in [1], and in most related techniques. Further, the signals in the STG are used as state variables in the synthesized circuit. Clearly, since the nodes in SG represent distinct states, enough "state information" must be contained in the state encoding in the synthesized circuit to be able to disambiguate distinct markings (states in SG). An STG has the unique state coding (USC) property if two distinct states in the state graph of the STG do not have identical binary codes.

Static and Dynamic Hazards. A static hazard is a 0→1→0 transition or a 1→0→1 transition in a signal where no transition for that signal should occur according to the specification.

A dynamic hazard is a 0→1→0→1 or 1→0→1→0 transition in a signal where a single positive (or negative) transition for that signal should occur according to the specification.

Valid Pairs. In analyzing the hazards in a circuit, it is typical to analyze the behavior when the inputs of the circuit change from an initial vector \( v_1 \) to a final vector \( v_2 \). Each such vector pair yields a transition cube where all the signals that can change value along a path from \( v_1 \) to \( v_2 \) in the state graph have an undetermined value "x", whereas the invariant signal components have the value in \( v_1 \).

Given an STG specification, the synthesis process yields a network for each logic function \( f_i \). In analyzing static hazards, the value of the \( i \)-th component is expected to be constant, hence the only vector pairs that need to be considered are those wherein \( f_i(v_1) = f_i(v_2) \). Each such pair is called a valid pair for signal \( i \).

3 Hazard Analysis and Avoidance

3.1 Previous Work

In the synthesis algorithm discussed in [3, 13], STGs are in essence converted into state graphs for finding a prime cover for a given output \( z \). Candidate state pairs, or valid pairs, with the same \( x \) values and the same values of the next state function \( f_x \), are formed. A transition cube is then determined for each valid pair and a check is made to see if this cube can only be partially covered by an on-set prime cover. A hazard is found if the transition cube is strictly partially covered as there exists a path in the transition cube that consists of subpaths of 0→1 or 0→1→0 values for \( f_z \). If \( f(i) \) denotes the time at which transition \( i \) occurs, the hazard condition is then identified as \( f(j) = f(i) < delay(i→x) - delay(j→x) \), where \( j \) and \( i \) are transitions causing the value of output \( z \) to be incorrect and correct respectively. The quantity \( f(j) - f(i) \) is named \( d3 \) and delay values are added independently (actually sequentially) to the value \( d3 \) associated with each hazard condition so as to preclude their occurrence. It is shown that such a delay insertion method is always feasible and satisfies all the hazard avoidance conditions.

Such methods do not operate directly at the STG level, and thus do not directly provide an intuitive explanation of the causes of hazards at the behavioral/STG level. They do not directly address dynamic hazards, as it is not easy to extend the basic concept to including transition cubes containing both on-set and off-set cubes for even hazard-free state pairs. In addition, the hazard avoidance procedure treats each hazard condition independently, since each condition is individually ensured by inserting a delay. If no attempt is made to use the slacks among conditions, the results from the procedure can be suboptimal. The linkage between the hazard-avoidance conditions and the system timing specifications was also not directly addressed.

3.2 Hazard Analysis: Overview

Since neither does a speed-independent design (a description at the STG level) guarantee a speed-independent implementation (at the gate level), nor are we interested in pure speed-independent designs and applications, hazard analysis has to be performed in order to have correct working asynchronous circuits. Further, it is desirable to perform such an analysis at the highest possible level.

We now delineate a method that directly operates on the STG, or the specification level. It uses a hazard analyzer for identifying the conditions that can potentially cause hazards, and produces a set of path delay (hazard avoidance) conditions under which the implementation will be hazard-free. The hazard analysis is based on a path-oriented approach for generating potential hazard candidates, both dynamic and static. We then simulate (evaluate) each candidate and compare its results with the correct results according to the specification. In addition to the benefits of working at a higher level of abstraction, this method handles some classes of dynamic hazards.

The hazard avoidance conditions generated by the analysis phase are then merged with the initial timing constraints in the specification to form a system of linear inequalities. An appropriate cost function is then defined and the entire linear inequality system is solved to yield an optimal solution using a linear programming procedure. A typical cost function is, for example, the
total number of delay elements inserted or the total delay along a specific path. The solutions are delays that each logic function has to have along a path; this can be implemented either by choosing an appropriate implementation from a library, or by inserting delay elements.

For many designs, this path-oriented approach provides a good visual way of determining potential hazards. By assisting in the identification of where and how hazards are possible at the STG level, it provides an easy visual technique for designers to assess a design and to make changes accordingly.

3.3 Generating Hazard Candidates

Given an STG, we next discuss a path-oriented technique that identifies a set of potential hazard conditions. This set of condition is pruned by a subsequent simulation phase that is detailed later.

Given an STG and a signal \( x \) in question, we first check to see if the STG has the USC property. If it does not, then the USC property is ensured by either inserting internal variables automatically or by asking the designer to modify the specification. Once an STG has the USC property, we can then simplify the graph by contracting the graph [1] such that the contracted/simplified graph consists of only the signal \( x \) and signals in the input set of \( x \). A signal \( y \) is in the input set of \( x \) if the signal \( y \) is a direct input to the logic generating module for the signal \( x \) in the final implementation. Techniques for determining the input set of a signal can be found in [7]. This simplification is useful, since the complexity of the following hazard checking procedure usually decreases significantly after the simplification.

For graphs where \( x \) only goes up and down once each, the contracted graph is then divided into two groups, group \( x^+ \) and group \( x^- \). Group \( x^+ \) consists of all signal transitions, together with their ordering relations, that can occur after \( x^+ \) and before \( x^- \) in the specification. Group \( x^- \) consists of all signal transitions that can occur after \( x^- \) and before \( x^+ \). For signals that go up and down more than once, one group is created for each adjacent \( x^+ \) and \( x^- \) pair. The following discussions will assume that signals only go up and down once. Multiple up-and-down transitions of a signal can be handled similarly.

All transitions in these two groups maintain the same order as in the contracted graph. For signals transitions ordered with respect to \( x^+ \) and \( x^- \), it is trivial to determine which group they belong to. The signals that are potentially concurrent with \( x^+ \) belong to both groups.

Consider, for example, the STG in Figure 1. For signal \( x \), group \( x^+ \) includes \( z^- \), and group \( x^- \) includes \( t^- \cdot z^+ \cdot t^+ \), where \( \cdot \) indicates sequential transitions. For signal \( y \), group \( y^+ \) includes \( x^- \cdot t^+ \cdot x^+ \cdot t^- \) and \( y^- \cdot y^+ \) includes \( t^- \cdot z^- \cdot t^+ \). For signal \( z \), group \( z^+ \) includes \( y^- \cdot t^+ \cdot z^+ \cdot y^+ \) and group \( z^- \) includes \( (z^- \cdot t^-) \cdot y^+ \), where \( \cdot \) indicates concurrent transitions. For signal \( t \), group \( t^+ \cdot t^- \) includes \( x^+ \cdot (z^- \cdot t^-) \cdot y^+ \), and group \( t^- \cdot t^+ \) includes \( y^- \cdot t^+ \cdot y^+ \).

Given the above groups, we can now motivate the conditions that can lead to hazards.

**Definition 1 (Inverted Signal Ordering)** An ordering of the signals in either of the groups \( x^+ \ldots x^- \) or \( x^- \ldots x^+ \) that is different from the specified order is said to be inverted.

Observe that a hazard can potentially occur at signal \( x \) only if some signal transitions in either the group \( x^+ \ldots x^- \) or the group \( x^- \ldots x^+ \) are out of order in time when they reach the logic element that computes \( x \), i.e., if ordering of the signals is inverted. A potential hazard condition is said to occur when an inverted ordering can lead to a hazard for output \( x \).

The hazard conditions in [2] are related to the notion of valid pair of input vectors. Recall that a valid pair in the context of static hazards is a pair of states reachable from each other (i.e., having a path between them in the state graph) and having the same \( f_x \) and \( f_y \) value. In comparison, in the definition of a hazard condition given above, the corresponding notion of valid pairs is a pair of states reachable from each other but that do not necessarily need to have the same \( f_x \) (i.e., next-state) value. It is easy to see that this hazard condition is weaker than the condition that just detects static hazards.

Consider the example shown in Figure 2. Here, \( x^- \ldots x^+ = \{ y^+, z^-, y^- \} \) and \( x^- \ldots x^+ = \{ z^+ | y^- \} \). Since \( y^+ \) and \( z^- \) are in group \( x^+ \ldots x^- \), an inverted order for \( y^+ \) and \( z^- \) produces a hazard for \( x \). Similarly, since \( z^+ \) and \( x^- \) are in both \( y^+ \ldots y^- \) and \( y^- \ldots y^+ \) if the transitions \( z^+ \) and \( x^- \) are inverted, they produce two hazards for \( y \), one when \( y = 0 \), and one when \( y = 1 \). There are no hazards for \( z \). This is consistent with the analysis of static hazards in [3] for this example.

This perspective on the origin of hazards is intuitively easy to understand. A synthesized circuit is designed to achieve the behavior of the "correct" STG description. If signals \( a \) and \( b \) are inputs to (the logic implementing) signal \( x \), with transition \( a^n \) preceding transition \( b^m \) in the specification, the circuit will perform correctly if the assumption holds in the actual implementation. However, since the delays of each gate and wire are different, it is possible that \( a^n \) traverses a slower route than \( b^m \) to reach \( x \), thus violating the assumption in the specification. This situation causes hazards if the sequences of transitions \( a^n, b^m \) and \( b^n, a^m \) generate different \( x \) output values. If \( t(a^n) \) denotes the time at which the transition \( a^n \) occurs, and \( d(a^n\rightarrow x^n) \) denotes the delay between \( a^n \) and \( x^n \), then \( t(b^m) + d(a^n\rightarrow x^n) \) must be less than \( t(b^m) + d(b^n\rightarrow z^n) \). Thus, the condition required to prevent a hazard from occurring is \( t(b^m) - t(a^n) > d(a^n\rightarrow x^n) - d(b^n\rightarrow z^n) \). The task of implementing a hazard free circuit from an STG specification then involves solving the conjunctions of all such conditions.
3.4 Hazard Analysis By Simulation

The conditions that lead to an inverted signal ordering is not a sufficient condition for the presence of hazards. An inverted signal ordering will generate a hazard only if the logic implementation produces differing results for the original and inverted orderings.

In order to determine precisely when hazards can occur, the circuit is first synthesized from the given STG specification. This process yields a boolean equation for each output variable, and an initial circuit implementation is then evaluated to detect potential hazard conditions. Two variations in the evaluation technique can be used.

The first method is a straightforward approach. When an input transition $y^t$ to $z$ arrives, that new $y$ value is used in the boolean evaluation and a new $z$ value is computed. When the next input transition $z^t$ arrives, the new $x$ value and the new $z$ value are used for computing the next $z$ output value. This evaluation process then repeats until all input transitions in question are evaluated in sequence.

The second method does the same except that the original value at the beginning of computation is always used for computing any next $z$ values. This method detects all sequences that may potentially switch $z$ twice, assuming that the memory device for implementing $z$ does not switch fast enough, e.g., like an ideal flip-flop; such behavior leads to static hazards. This second method, together with the restriction that at least one of the transitions immediately preceding $z$ is always the last transition in an inverted signal ordering, makes this approach to be equivalent to the static hazard analysis in [3].

Note that in general the feedback loop for a variable can have an arbitrary delay in the range $[0, \infty]$. This entire range of delays needs to be considered to detect all of the possible hazards. However, in practice, the delay is usually in a bounded range $[\min, \max]$ and this can be used to significantly reduce the computation complexity for the full blown analysis. Also note that the second method is weaker than the first one for detecting static hazards, as the first method may record the switch and detect no further change of value of $z$. The second approach thus depends on the design of the flip-flops used in the circuit. If the flip-flop internal feedback time is negligible through proper buffering before the final output, then the first method is a reasonably good method of detecting any hazard. Otherwise, both methods should be used together and the union of the detected hazards are reported. For simplicity, only the first method will be used in the discussion of the remainder of this paper.

The method can be summarized as follows.

0. For each variable $z$, contract the STG.
1. Synthesize the logic for $z$: this yields a boolean equation and an associated two-level implementation for $z$.
2. Construct $z^+ \ldots z^-$ and $z^- \ldots z^+$.
3. Evaluate the value of the output $z$ for each inverted signal ordering, and compare the result with the correct output. If the results differ in number or parity of signal transitions, a hazard is detected.

For a posteriori design analysis, the number of inverted signal orderings can be large in the worst case ($O(n!)$. In practice, the number of input signals involved for a given signal output is usually quite small, and this computation may not pose a problem in typical control circuits. For synthesis, very few inverted signal orderings can lead to hazards; further, these cases can be identified during synthesis, as discussed in a forthcoming report.

3.5 An example

The following example illustrates the detection of dynamic hazards by this procedure. In Figure 1, if we restrict ourselves to static hazards for the signal $z$, when $z = 1$ (i.e., the states after $z^+$), the only valid pair is 0110 and 0011, with the second and fourth bit representing $y$ and $t$ respectively. The transition cube $z^+ - z^-$ is covered by a prime implicant $z$ and therefore there is no hazard. Another way of verifying this is to note that for both orderings $y^t$ and $t^y$, the output of $z$ stays at 1 and there is thus no static hazard. However, if $t^+$, $z^+$, and $y^+$ are out of order, and the order becomes $y^t$, $z^+$, $t^y$, $y^+$ when they reach $z$, the transitions associated with $z$ become 1−0→1−0 as opposed to the expected sequence 1−1→0−0. This indicates a dynamic hazard, which is associated with the hazard condition $t(y^t) - t(t^y) < d(t^y - z^+) - d(y^t - z^-)$.

4 The Hazard-Avoidance Procedure

In a design specification, timing information can be incorporated by describing the constraints on the delays between various pairs of transitions for correct asynchronous operations. Such timing constraints are usually guaranteed either by (1) the environment that a circuit operates in or by (2) the synthesizer, that views the timing information as being mandated by the environment and then ensures that they are met by the synthesized circuit.

The hazard-avoidance conditions and the original timing constraints form a system of inequalities involving delays between transitions. Linear programming techniques can be used to solve the system, once a linear objective function is defined.

In the most general case, each wire of a electrical net can have a different delay and each I/O path through a simple/complex gate can have a different delay. We can bundle the delays of a wire and I/O path together. The resultant circuit model we use has delays for each wire and zero delay for gates, as shown in Figure 3. The hazard analysis method discussed here can be intuitively explained in the context of such a delay model. Since the internal gate delays are zero, the only way hazards to occur is for the input signals to arrive in an order different from the specification. In such cases, delays have to be added to ensure that the correct order is restored.

Given an initial circuit implementation, the procedure below yields a set of inequalities embodying the hazard free conditions. The function $\text{ica}(s_i, s_j, s_k)$ returns the set of least common ancestors of $s_i$ and $s_j$ that lead to $s_k$.

![Figure 3: Delay model.](image-url)
Figure 4: (a) Example STG (b) contracted STG for Lr.

Figure 5: (a) An initial implementation (b) timed.

Hazard-free-conditions \(\leftarrow \emptyset\)
for each output \(s_k\) causing hazards
for each node \(e\) in \(\text{lca}(s_i, s_j, s_k)\)
Hazard-free-conditions \(\leftarrow\)
\(t(s_j) - t(s_i) > d(s_i \rightarrow \text{node}_e) - d(s_j \rightarrow \text{node}_e)\)

4.1 An Example

We will illustrate the hazard avoidance procedure with an example used in [1] and [2]. Figure 4 shows the STG description; a circuit initially synthesized from this STG is shown in Figure 5(a). We will assume that each inverter has a delay in the range \([2,5]\), i.e., an inverter has a delay that is a minimum of 2 units and maximum of 5 units, and that all other gates have a delay in the range \([3,7]\).

In order to enable a comparison with earlier methods e.g., in [2], we will here consider only static hazards. Note that a dynamic hazard is possible for \(Lr\) (Figure 4(b)), because the loop corresponding to \((Sr^+, Sr^-)^*\) can be traversed any number of times before \(Cr^+\) is enabled. If the path from \(Sr^+\) to \(Lr\) involves an extraordinary amount of delay and therefore \(Cr^+\) causes \(Lr^-\) before all the \(Sr^-\), \(Sr^+, Sr^-\), \(Sr^+\) \(\ldots\) \(Sr^+\) \(\ldots\), \(Sr^-\) transitions reach \(Lr\), \(Lr\) will undergo the transitions \(0\rightarrow 1\rightarrow 0\rightarrow 1\rightarrow \ldots\) as many times as the signal \(Sr\).

By contracting the STG and synthesizing the logic, we get \(Lr = Sr + Cr^+La, Ca = Cr^+La\) and \(Sa = Sr^+La\). In the group \(Lr^+ \ldots Lr^+\) for signal \(Lr\), a hazard occurs when \(La^-\) and \(Cr^-\) are out of order when they reach \(Lr\), i.e., \(Cr^-\) effects \(Lr\) before \(La^-\). The condition for avoiding such a hazard is

\[
(t(Cr^-) - t(La^-) > d(La^- \rightarrow a3^-) - d(Cr^- \rightarrow a3^+). \quad \ldots(0)
\]

Note that the two paths join at \(a3\); at this stage, one of the two signal will dominate and propagate further. Since \(La^-\) precedes \(Cr^-\) (Figure 5), following the path from \(La\) to \(Ca\) through the logic, we obtain \(t(Cr^-) = t(La^-) + d(i2^+) + d(i2^+ \rightarrow Ca) + d(Ca^+ \rightarrow Cr^-)\), where \(d(i2^+)\) denotes \(d(i2^- \rightarrow 2^+2^-)\). We can then rewrite (0) as

\[
d(i2^+) + d(i2^+ \rightarrow Ca) + d(Ca^+ \rightarrow Cr^-) > d(La^- \rightarrow a3^+) - d(i1^+) - d(i1^+ \rightarrow a3^+) \quad \ldots(1)
\]

In the group \(Lr^+ \ldots Lr^+\), a hazard occurs when \(La^+\) and \(Sr^-\) are out of order. The hazard avoidance condition is

\[
t(Sr^-) - t(La^+) > d(La^+ \rightarrow Lr^-) - d(Sr^- \rightarrow Lr^-)
\]

and it can be similarly expanded into

\[
d(La^+ \rightarrow Sa^+) + d(Sa^+ \rightarrow Sr^-) > d(La^+ \rightarrow a3^+) + d(a3^+ \rightarrow Lr^+) - d(Sr^- \rightarrow Lr^-) \quad \ldots(2)
\]

For signal \(Ca\), a hazard occurs when transitions \(La^+\) and \(Cr^+\) reach \(Ca\) out of order. The hazard avoidance condition is

\[
d(La^+ \rightarrow Sa^+) + d(Sa^+ \rightarrow Sr^-) + d(Sr^- \rightarrow Sa^-) + d(Sa^- \rightarrow Cr^-) > d(i2^+) + d(i2^- \rightarrow Ca^-) - d(Cr^+ \rightarrow Ca^-) \quad \ldots(3)
\]

There is no hazard for signal \(Sa\).

In order to make the system solvable, we insert delays for each wire and each gate output before fanout if the number of either fanin or fanout is greater than one. We then have,

\[
D1 + d(i2^+) + D2 + d(i2^+ \rightarrow Ca^+) + D2a + d(Ca^+ \rightarrow Cr^-) + D4 > D7 + d(La^- \rightarrow a3^-) \quad \text{D14 - D5 - d(i1^+) - D6 - d(i1^+ \rightarrow a3^+) - } \quad \ldots(4)
\]

\[
D10 + d(La^+ \rightarrow Sa^+) + D10a + D12 + d(Sa^+ \rightarrow Sr^-) + D7 + d(La^- \rightarrow a3^+) + D6a + D8 + d(a3^+ \rightarrow Lr^-) - D13 - D9 - d(Sr^- \rightarrow Lr^-), \quad \ldots(5)
\]

\[
D10 + d(La^+ \rightarrow Sa^+) + D10a + D12 + d(Sa^+ \rightarrow Sr^-) + D13 + D11 + d(Sr^- \rightarrow Sa^-) + D10a + D15 + d(Sa^- \rightarrow Cr^-) > D1 + d(i2^-) + D2 + d(i2^- \rightarrow Ca^-) - D14 - D3 \quad - d(Cr^+ \rightarrow Ca^-). \quad \ldots(6)
\]

Note that delays have been added to both the output side and the input side of a gate. Strategies can be developed such that fewer variables are used if the cost function used can be defined on only those variables.

If system timing constraints on the external signals exist, they are treated the same way as the above inequalities. There is no external system timing specified
for this example. We will assume that \(d(Ca^+\rightarrow Cr^-)=0\), \(d(Sa^+\rightarrow Sr^-)=0\), \(d(Sa^-\rightarrow Cr^+)=0\). Further, we use the lower bound values for the left-hand side terms and negative right hand side terms, and upper bound values for positive right hand side terms and negative left hand side terms. From (4) (5) (6), we have,

\[
\begin{align*}
D1 + D2 + D2a + D4 &> D7 - D14 - D5 - D6 - 3 \\
D10 + D10a + D12 &> D7 + D6a + D8 - D13 - D9 - 8 \\
D10 + 2*D10a + D12 + D13 + D11 + D15 &> D1 + D2 - D14 - D13 + D15 - 3.
\end{align*}
\]

A simple cost function (to be minimized) can be the sum of total delay value, i.e., sum of \(D_i\)'s, as this may approximate the additional area needed and the overall slow-down once the delay elements are inserted. A cost function may also consider the total number of delays inserted, the slacks (the safety margin) and/or the delay along a certain path.

For this example, we use \(\sum D_i\) as the cost function. A possible solution is \(D10a=9\), which says that if gate \(g_2\) is slowed down by 9 units, the circuit is hazard-free. Note that this implementation happens to eliminate the dynamic hazard, caused by \(Cr^+\) reaching \(Lr\) earlier than \(Sr^-\), in this particular example. In comparison, [2] requires a delay to be added to each inequality independently. Thus their solution is \(D13=9\) and \(D14=4\). The total delay is 13 and consists of two delays, both of which have to be delay buffers. Moreover, their solution increases the delay along the longest path \(Cr^+\rightarrow Lr\).

Note that it is usually more costly to use buffers as delays than to slow down a gate. For circuit implementations that employ a cell library, several implementations having different delays for the same functionality are available, and it may not be necessary to use delay elements for gate output delays. We can use the fastest available gates as the initial solution to such a system, and then obtain needed delays by solving the system. Slower gates approximating the solution can then be used instead of inserting delay buffers, as long as the approximation still satisfies the inequalities.

Further, if certain specific design styles or constraints, for example, similar I/O path delays for a single gate, isochronic forks, etc., are known, these constraints can be added to the system to simplify the problem and to restrict the solution domain. Tools can be written to provide designers options to tailor the inequality system to their specific design environments.

5 Experimental Results

The appropriateness of a particular implementation is strongly dependent on the constraints imposed by the environment that a circuit is designed to operate in. This information is captured by the initial timing specifications and restrictions. Note that, unlike synchronous designs, the longer delay between flip-flops may be on a path that is non-critical for asynchronous operations. In the experiments below, a simple criterion, consisting of the sum of total inserted delays, is used. This provides a measure of how much the circuit will be slowed down overall and how much additional area is needed, in the absence of environmental timing specifications. The results for a set of STG examples in the literature are shown below. The method used in [2] are also included for comparison. "No. of dynamic hazards" is the number of hazards detected by the current method.

<table>
<thead>
<tr>
<th>Ckt. name</th>
<th>No. of dynamic hazards</th>
<th>No. of delays [2]</th>
<th>No. of delays</th>
<th>Total delay [2]</th>
<th>Total delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>C133</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>C150</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C172</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>xyz</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xyz</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

6 Conclusions

The primary objective of this paper was to explore a technique that provides a designer an intuitive (visual) correlation between specifications at the level of signal transition graphs and conditions that lead to hazards. The motivation was to provide insight into how the STG specifications should be altered, if necessary. This led to a path-oriented approach for hazard analysis. In essence, inverted signal orderings in an implementation can potentially lead to hazards, but this set is pruned by simulation in order to detect those conditions that can actually lead to hazards. We have also described a technique for avoiding hazards by adjusting the delays in a circuit; rather than doing this piecewise for each path, we have shown how improved results can be obtained by solving an appropriately formulated linear programming problem. In addition to the benefits of working at a higher level of abstraction, this method handles some classes of dynamic hazards (under bounded delay assumptions) in addition to static hazards.

References