# Removing CSC violations in asynchronous circuits by delay padding

K.-J. Lin
C.-S. Lin

*Indexing terms: Asynchronous circuits, Signal transition graphs*

**Abstract:** A novel alternative for removing CSC (complete state coding) violations in asynchronous circuit synthesis for STGs (signal transition graphs) is presented. The main feature of the work is to exploit delays in the physical circuit to remove CSC violations. Its main advantages are that it (i) does not need to obey the noninput constraint and (ii) saves area overhead when a CSC violation in the state graph does not actually appear in the physical circuit. The delay constraint for removing each CSC violation is formulated. Then an algorithm is proposed to derive a consistent set of constraints to ensure that all violations are removed. If a consistent set exists, it is shown that those constraints can always be satisfied by padding delays during hazard analysis, and therefore hazard-free circuits without any CSC violation can be derived. Based on this approach, the marked-graph benchmarks, hitherto unsolvable due to the noninput constraint in existing methods, are now resolved.

## 1 Introduction

Asynchronous design has received much attention in recent years. A considerable number of proposals have been made to automatically synthesise hazard-free asynchronous circuits starting from newly developed event-based or state-based specifications [1]. In the existing methods, a prerequisite for realising circuits is to satisfy the so-called CSC (complete-state-coding) property. Violation rectification must be completed before circuit realisation. To satisfy such a property, arcs (for removing states) and/or internal state signals are inserted into the original specification without changing the observable behaviour of the environment [2–5]. Generally, signal insertion can deal with a wider range of violations, while arc insertion may save area.

However, there are two main drawbacks in the existing methods for removing CSC violations: (i) the inserted arcs and signals must not be connected to an input transition, that is, the insertion must obey the noninput constraint; some benchmark examples, therefore, cannot be solved under such a constraint; (ii) even though violations exist in the state graph, those problematic states may not appear in the physical circuit because of the inherent delay, hence, the area and/or performance overheads to remove such physically nonexistent CSC violations are wasted.

In this paper we propose a novel alternative for removing CSC violations in asynchronous circuit synthesis for STGs under the bounded delay model. The key idea is to pad delays selectively to slow down some transitions in order to render some problematic states unreachable in the physical circuit. We first formulate the constraint for delays to remove each CSC violation. Then an algorithm is proposed to derive a consistent set of constraints to remove all violations. If a consistent set exists, we show that those constraints can always be satisfied by padding delays during hazard analysis and therefore hazard-free circuits without any CSC violation can be derived. The proposed approach is not burdened with the two main drawbacks of the existing methods. First, the delay padding can be used to delay the input signal to affect the outputs, since this just slows down circuit response while it does not change the causal relations among signal transitions. Therefore, the delay padding does not need to obey the noninput constraint. Secondly, it is possible that no delay padding is needed when CSC violations in the state graph do not actually appear in the physical circuit. Based on this approach, we have been able to resolve those unsolvable marked-graph benchmarks due to the noninput constraints in the existing methods.

A related work is [6], which pointed out the unreachability of some legal states in timing STG. Those unreachable states reduce the possibilities of CSC violation and also reduce the implementation area. However, their work needs predefined timing among signal transitions and thus constrains the subsequent synthesis work. Nevertheless, this predefined timing information may not be satisfied after hazard removal. Iterations between CSC satisfaction and circuit implementation may occur. In addition, it did not provide any method to remove CSC violations.

## 2 Preliminaries

A signal transition graph, STG, can be viewed as an interpreted Petri net in which each transition is interpreted as a physical signal transition of asynchronous behaviour [7]. A Petri net (PN) is a 4-tuple $N = < P, T,$



*IEE Proceedings* online no. 19960634

Paper first received 4th December 1995 and in revised form 14th May 1996

The authors are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, Republic of China

$F, m_0 >$, where $P$ is a set of places, $T$ is a set of transitions, $m_0$ is the initial marking and $F \subset (P \times T) \cup (T \times P)$ is the flow relation. A place $p$ is a fanin of a transition $t$, and $t$ is a fanout of $p$, if $(p, t) \in F$. Conversely, a transition $t$ is a fanin of $p$, and $p$ is a fanout of $t$, if $(t, p) \in F$. The execution of a PN starts from the given initial marking $m_0$. A marking is a set of tokens upon a set of places. A transition is enabled when all its fanin places carry tokens. Then it can be fired. The firing of a transition is to remove one token from each of its fanin places and add one token to each of its fanout places. After a transition is fired, a new marking is reached. A marking $m$ is called live if each transition can always be enabled in some marking reachable from $m$. A net is live if its initial marking is live. A marking $m$ is called safe if no place can ever carry more than one token after any sequences from $m$. A PN is safe if its initial marking is safe. Under our current consideration, the STG class has an underlying live and safe marked graph, in which each place has at most one fanin and at most one fanout. For simplicity, places are removed, that is, each arc in an STG originally carries a place. Furthermore, we restrict ourselves to strongly connected nets. This means the considered STG can execute infinitely.

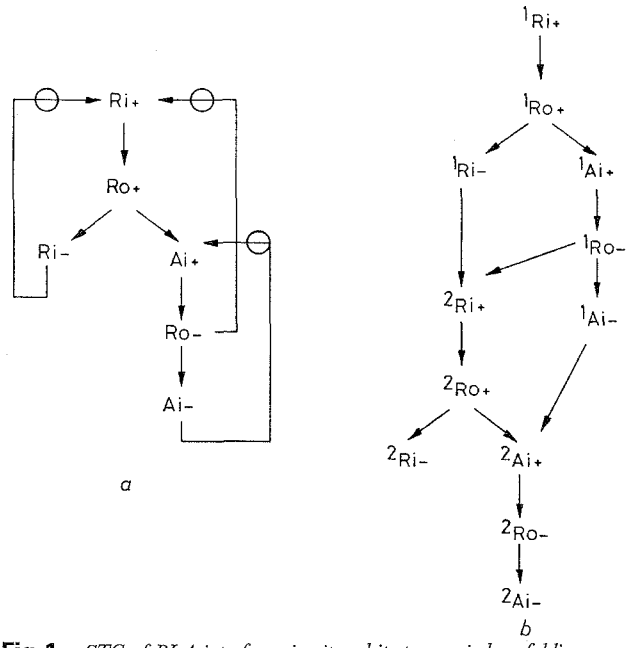


**Fig. 1** *STG of PLA interface circuit and its two-periods unfolding*
*a* STG; *b* two-period unfolding

The STG naturally captures the characteristics of a general timing waveform. A simple STG from [2] is shown in Fig. 1*a*. The transitions of a signal $x$, denoted by $x+$ and $x-$, are the rising ($0 \rightarrow 1$) and falling ($1 \rightarrow 0$) transitions, respectively. Henceforth, $x^*$ will denote a certain transition of signal $x$ (i.e. either an $x+$ or an $x-$) and $\overline{x^*}$ will denote its inverse transition (i.e. either an $x-$ or an $x+$). The signal behaviour described in the STG is according to the Petri net firing rules. If two transitions can be enabled at the same time, they are concurrent. Otherwise they are ordered. In an STG, an arc between two ordered transitions may be redundant. An arc $x_i^* \rightarrow x_j^*$ is not redundant only if there exists a marking in which all input arcs to $x_j^*$ except this carry tokens. For such a marking, if $x_i^*$ is fired, $x_j^*$ can be enabled immediately. We call $x_i^*$ ($x_i$) an enabling transition (signal) of $x_j^*$. In this paper we assume that all redundant arcs in the STG have been removed.

The STG has an equivalent finite-state-machine representation called a state graph (SG). The SG is a directed graph, in which each vertex (i.e. a state) is in one-to-one correspondence with a marking reachable from the initial marking, and each arc $s_1 \xrightarrow{t^*} s_2$ represents that $t^*$ is enabled in $s_1$ and $s_2$ can be reached from $s_1$ through the firing of $t^*$. The SG represents STG concurrency as an interleaving of transitions. That is, in each state concurrently enabled transitions can be fired in any order, but only one is fired at a time to reach a new state. Note that actual orders among signals in the physical circuit may not completely follow the sequence of the state graph due to the circuit delays. Since the underlying net of the considered STG is live, safe and strongly connected, the SG is strongly connected. The SG of the STG in Fig. 1*a* is shown in Fig. 2. The SG captures the state of all signals (input, output and internal signals) in a circuit. As the total-state model in the classical asynchronous design, all signals are considered as state variables, and their Boolean values are used to encode states. For each $s_1 \xrightarrow{x^*} s_2$ ($s_1 \xrightarrow{\overline{x^*}} s_2$) arc, the value of $x$ in the coding is 0 (1) in $s_1$ and 1 (0) in $s_2$, while all other signals must have the same value in both states. To ensure that the state assignment is consistent, a transition $x+$ ($x-$) can be enabled only in a state whose code for $x$ is 0 (1). The following requirement is proposed to have a consistent state assignment by [7].

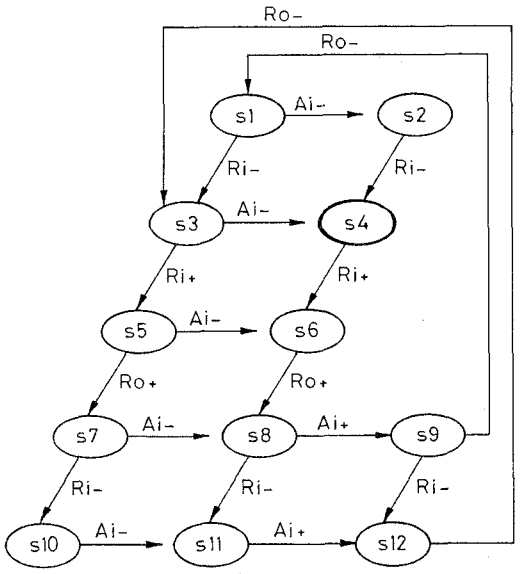


**Fig. 2** *SG of STG of PLA interface circuit*

*Definition* (*liveness*) [7]: An STG is live iff

(i) the underlying Petri net is live and safe

(ii) for each signal $a$, $a+$ and $a-$ occur alternately, and no two transitions of $a$ are concurrent.

Fig. 3 shows the SG with a consistent state assignment. From the consistently encoded SG of an STG, we can define the output function (or next-state function in [8]) of each noninput signal. For each noninput signal, if its transition is enabled in a state, then its output function has a different Boolean value from the state code. Otherwise, it is the same as the state code. All signals are the input variables to the output function. The current state code corresponds to an input vector to excite the output function. The output function for *Ro* in Fig. 1*a* is also shown in Fig. 3. In a live STG, two different states may have the same state code. To ensure each state code (input vector) to predict a

deterministic output value, the following property is required.

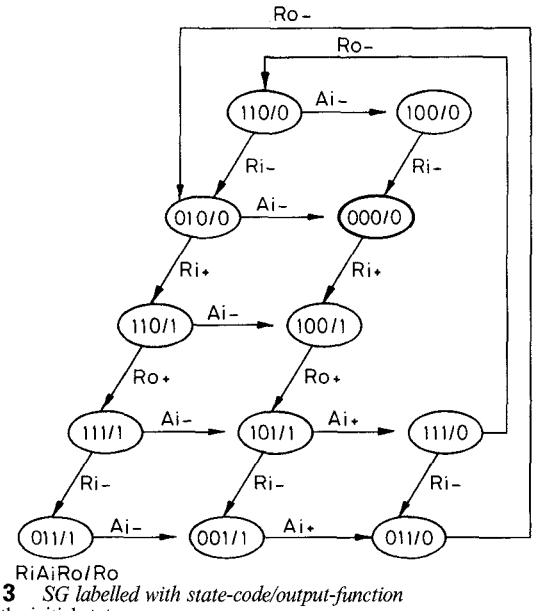


**Fig. 3** *SG labelled with state-code/output-function*
*s*4 is the initial state

*Definition* (*complete state coding* (*CSC*)): A live STG has a CSC property iff any two states which enable different sets of noninput signal transitions have distinct state codings.

Fig. 3 gives examples of CSC violations. Both states *s*1 and *s*5 are assigned the same binary label 110 but *s*5 enables the output transition *Ro*+ while *s*1 does not (thus *s*5 predicts *Ro* = 1 while *s*1 predicts 0). Clearly, we cannot have a logic implementation which accepts the same state code but produces different outputs. The main concern of this paper is to remove CSC violations.

## 3 Delay arc

Our key idea to remove a CSC violation is to force a pair of concurrent transitions to affect outputs in some definite order such that problematic states cannot be reached in the physical circuit. This constraint can be represented as a delay-arc between the transition pair in the original STG. The state sequence (firing order of transitions) is expected to follow those added delay arcs, while the enabling order is not changed. In the physical circuit, a constraint is achieved by inherent delays or padded delays to slow down a transition related to the other one. In Section 5 we show how to achieve the constraint during hazard analysis. This Section will show how to find a delay arc for removing a certain problematic state. Because of the different types of concurrent behaviour, the effect of a delay arc between two concurrent transitions may be different. We will summarise the kinds of arcs that are permissible for our problem.

It is important to note the difference between the regular arc used in the existing methods and the delay arc. A regular arc forces two originally concurrent transitions to be enabled in order, while a delay arc only forces them to fire in order. Therefore, the expected state graphs are both the same, but a regular arc changes the output function while a delay arc retains the original specification. Furthermore, since a regular arc changes the enabled order (i.e. changes the causal relation), it must not be directed to an input transition, unless modification of the input specification is allowed. On the other hand, the delay arc only delays the effect of an input transition passed to some outputs. This does not change the specification and hence is permissible.

We now show how to find a delay arc for removing a problematic state. To remove a state $s_i$, all possible firing sequences from the initial marking to $s_i$ must be removed. We check whether there exists one of the fanin transitions of $s_i$ which is concurrent with any of its fanouts. If such a transition pair exists, a delay arc can be inserted between them to ensure that, from each state enabling both of them, the fanin transition always fires after that fanout transition, and therefore all the state sequences to reach $s_i$ will not occur. Otherwise, if $s_i$ has only one predecessor state, we try to find such pairs from its predecessor state (our current method discards the search if $s_i$ has more than one predecessor state). For instance, if we want to remove $s_1$ in Fig. 3, the pair (*Ri*–, *Ro*–) is found and arc *Ri*– → *Ro*– is inserted to attain our goal. However, in certain conditions, an arc added between two concurrent transitions may be redundant (does not remove any state) or may render the STG nonlive, which clearly is not allowed. The different effects of arcs result from the different types of concurrent behaviour. The following will show the classification of concurrent behaviour, which is a basis to select correct delay arcs in the following Section. The classification in fact is also applicable to the regular arc for removing states. To the best of our knowledge, no previous work has ever addressed this classification.

The classification for concurrent behaviour is based on the impact of an added arc between two concurrent transitions. Since the impact depends on the placement of the initial marking, we exploit the two-period unfolding of an STG for the analysis. The unfolding has been used to analyse the transition relation in an efficient way [9]. It can be seen as an acyclic process in which each transition corresponds to a single instantiation of a transition in the original STG. Let $x^*$ be a transition in the original STG. We denote its $k$th occurrence in the unfolding as ${}^kx^*$. It was proved that the first two periods of the unfolded STG are enough to derive all precedence relations in $O(N^3)$ complexity with respect to the number of transitions $N$. Fig. 1*b* shows the unfolding of Fig. 1*a*. Henceforth, we use $\Rightarrow$ to denote the partial-order relation and $\|$ the concurrent relation in the unfolding of an STG. We now recall a result from [9], that two transitions $t_i$ and $t_j$ are concurrent in an STG iff there exist ${}^{k_1}t_i$ and ${}^{k_2}t_j$ such that ${}^{k_1}t_i \| {}^{k_2}t_j$ in the unfolding of the STG. We then classify the concurrent relations to four types as shown in the Table 1. The arcs $t_i \rightarrow t_j$ in the first three types and arc $t_j \rightarrow t_i$ in type 1 are permissible. The arc $t_i \rightarrow t_j (t_j \rightarrow t_i)$ removes those states with $t_j$ as a fanin and $t_i$ as a fanout ($t_i$ as a fanin and $t_j$ as a fanout). The arc $t_j \rightarrow t_i$ in type 2 is redundant, because from the initial marking the relation ${}^1t_j \Rightarrow {}^1t_i$ already exists, while it does not force the required relation ${}^2t_j \Rightarrow {}^1t_i$. For example, for removing $s_4$, we find the arc *Ri*+ → *Ai*–. However, it cannot force *Ri*+ to fire always before *Ai*– in $s_3$ and thus cannot remove *s*4. Hence, such an arc is not permissible. In type 3, the arc $t_j \rightarrow t_i$ is also not allowed. Because $t_i$ may fire twice while $t_j$ keeps enabled, such an arc is not enough to force them to be

ordered. For example, even added to $Ri- \rightarrow Ai-$, the two transitions $Ri- \rightarrow Ai-$ are still concurrent. Actually, this arc renders the STG unsafe. As for type 4, with the same reasoning as type 3, no arc is permissible. In our procedure, all added delay arcs belong to those permissible arcs in Table 1.

**Table 1: Classification of concurrent behaviour**

| Types | Conditions | Permissible arcs | Examples in Fig. 1 |
|---|---|---|---|
| 1 | ${}^1t_i \parallel {}^1t_j \cap {}^1t_i \Rightarrow {}^2t_j \cap {}^1t_j \Rightarrow {}^2t_i$ | $t_i \rightarrow t_j$ or $t_j \rightarrow t_i$ | (*Ri–,Ai+*) |
| 2 | ${}^1t_j \Rightarrow {}^1t_i \cap {}^2t_j \parallel {}^1t_i$ | $t_i \rightarrow t_j$ | (*Ai–,Ri+*) |
| 3 | ${}^1t_i \parallel {}^1t_j \cap {}^1t_i \parallel {}^2t_j \cap {}^1t_j \Rightarrow {}^2t_i$ | $t_i \rightarrow t_j$ | (*Ai–,Ri–*) |
| 4 | ${}^1t_i \parallel {}^1t_j \cap {}^1t_i \parallel {}^2t_j \cap {}^1t_j \parallel {}^2t_i$ | No | |

$\parallel$: concurrent relation. $\Rightarrow$: ordered relation. $\cap$: and-operation

## 4 Consistent delay-arc set

Requirements must be satisfied when more than one delay arc is needed to remove all CSC violations, to retain the observed behaviour of the environment and to ensure that the relations specified by the selected arcs can be met in the physical circuit and a hazard-free circuit can be derived. We firstly formulate those requirements. Then, we propose an algorithm to derive a consistent set of delay arcs, i.e. a set satisfying all requirements, for removing all CSC violations.

The first requirement for a consistent set is formulated as follows.

*Liveness_and_environment_requirement*: Let $G$ be a live STG with initial marking $m_0$ and $G'$ be the one modified from $G$ by adding with a set of arcs: (i) all added arcs are restricted to the permissible arcs of the three types of concurrent behaviour in Table 1, and (ii) each directed cycle in $G'$ carries at least one token when given the initial marking $m'_0$ which is composed of all tokens in $m_0$ and one token for each inserted permissible arc of type 2.

The initial token placement for $G'$ is intended for STG liveness. Since $t_j$ is enabled before $t_i$ from $m_0$ in the type 2 concurrent behaviour, the underlying net cannot be live without a token on the arc $t_i \rightarrow t_j$.

The following theorem shows the desirable property from the requirement.

*Theorem 1*: If $G'$ satisfies the liveness_and_environment_requirement, it is live and does not change the behaviour observed by the environment (see proof in Appendix 8.1).

The second requirement for a consistent set is to ensure that all specified relations by the set can be met in the later hazard-removal step. For a delay arc $t_i \rightarrow t_j$ we need to perform timing analysis to check if the order is met in the physical circuit. If required, an actual delay value is determined and padded to slow down $t_j$. Although these tasks can be performed exactly only after physical circuits are realised, from the STG we can derive a set of transitions whose implemented circuits dominate the timing analysis. This prerealisation information can help us to check beforehand if the delay arc can be met after circuit realisation by delay padding. The following definitions introduce such a set of transitions.

*Definition* (*reference-cut set*): A set of transitions $R$ in a live STG is a reference-cut set with respect to a delay arc $t_i \rightarrow t_j$ if (i) $R$ cuts all directed cycles containing $t_i$, (ii) all transitions of $R$ are concurrent and (iii) each transition in $R$ is ordered with both $t_i$ and $t_j$.

We define the following assuming that each individual transition of a signal can be slowed down separately in the circuit implementation.

*Definition* (*reference set*): Let $R$ be a reference-cut set with respect to a delay arc $t_i \rightarrow t_j$. Given $R$, the reference set with respect to $t_i \rightarrow t_j$ is composed of the $t_i$ and the set of transitions residing between $R$ and $t_i$. The reference set is called mimimal if no other set derived from other reference-cut sets with respect to $t_i \rightarrow t_j$ is its subset. The corresponding reference-cut set is also called mimimal.

Now we formulate the second requirement for a consistent delay-arc set. The corresponding circuit of the minimal reference set dominates the delay analysis for a delay arc. Whether the order of a delay arc $t_i \rightarrow t_j$ is met in the physical circuit can be determined by checking the delay difference between $t_i$ and $t_j$ starting from the corresponding circuit of the minimal reference-cut set. We need to calculate two bounds for two circuit paths: a lower bound on the delay from the minimal reference-cut set to $t_j$ and an upper bound on the delay from the minimal reference-cut set to $t_i$. It will be evident in the following Section that we must ensure that the upper bound is less than the lower bound. Hence, the circuit delay of each transition in the minimal reference set (involving the upper bound) must be fixed before considering delays padded to $t_j$. Such constraints among transitions constrain the evaluation-order among delay arcs (evaluating whether the specified relation of a delay arc is satisfied and delays are padded if it is not met). That is, if transition $t_q$ belongs to the minimal reference set of arc $t_i \rightarrow t_j$, then each arc pointing to $t_q$ must be treated before $t_i \rightarrow t_j$. Clearly, if such orders among a set of arcs are cyclic, the set may not be satisfied simultaneously in the physical circuit. Therefore, we have the following requirement.

**Select_Consistent_Arc_Set():**
(Given a live STG, its SG and the set of state-pairs $V$ which cause CSC violations. Initialise the arc-set $A$ to be empty.)
1. Foreach violation $v_i$ in $V$
   Find candidate arcs, $A(v_i)$;
   Let $A = A \cup A(v_i)$;
2. Foreach arc $a_k$ in $A$
   Let the set of violations removed by arc $a_k$, $V(a_k)$, be empty;
   Foreach violation $v_i$ in $V$
      if $v_i$ can be removed by adding $a_k$ to the STG, let $v_i \in V(a_k)$;
3. Foreach arc $a_k$ in $A$, find the minimal reference-cut set and minimal reference set;
4. Construct Evaluation_Order_Graph, *EOG*;
5. Find a minimal subset $A'$ of $A$ such that $\forall v_i, \exists a_k \in A' \Rightarrow v_i \in V(a_k)$, and the subgraph of *EOG* induced by the vertex_set representing $A'$ is acyclic;
6. Derive an evaluation order for $A'$;/* i.e. a topological order in the *EOG* */
7. Return the $A'$ and the evaluation order.

**Fig. 4** *Algorithm to select a consistent set of delay arcs which remove all CSC violations*

*Acyclic_requirement*: The evaluation_order among the selected delay arcs cannot be cyclic.

Fig. 4 presents an algorithm which selects a consistent delay arc set for removing all CSC violations, i.e. a set which satisfies the liveness_and_environment_requirement and the acyclic_requirement. The first step finds candidate arcs for removing each violation according to the rules described in the preceding Section. For each conflict state pair, we can remove any of them or both them. Furthermore, a state may have more than one pair of concurrent fanout and fanin transitions. Hence, the

**Table 2: CSC violations and their corresponding candidate arcs**

| CSC violations | | Candidate arcs | | | | | |
|---|---|---|---|---|---|---|---|
| Items | State pairs | Items | Arcs | $S(a_k)$ | $V(a_k)$ | Min-cut | Min-ref |
| $v_1$ | $(s_1, s_5)$ | $a_1$ | $Ri- \rightarrow Ro-$ | $s_1, s_2$ | $v_1, v_2$ | $Ro+$ | $Ri-, (Ri+)$ |
| | | $a_2$ | $Ai- \rightarrow Ri+$ | $s_5, s_7, s_{10}$ | $v_1, v_3, v_4$ | $Ro-$ | $Ai-, (Ai+)$ |
| $v_2$ | $(s_2, s_6^+)$ | $a_1^*$ | $Ri- \rightarrow Ro-$ | $s_1, s_2$ | $v_1, v_2$ | $Ro+$ | $Ri-, (Ri+)$ |
| $v_3$ | $(s_7, s_9)$ | $a_3$ | $Ai- \rightarrow Ro+$ | $s_7, s_{10}$ | $v_3, v_4$ | $Ro-$ | $Ai-, (Ai+)$ |
| | | $a_4$ | $Ri- \rightarrow Ai+$ | $s_9, s_1, s_2$ | $v_1, v_2, v_3$ | $Ro+$ | $Ri-, (Ri+)$ |
| $v_4$ | $(s_{10}, s_{12})$ | $a_5$ | $Ro- \rightarrow Ri-$ | $s_{12}, s_{11}, s_{10}$ | $v_4$ | $No$ | |
| | | $a_3^*$ | $Ai- \rightarrow Ro+$ | $s_7, s_{10}$ | $v_3, v_4$ | $Ro-$ | $Ai-, (Ai+)$ |

$S(a_k)$ = set of states removed by $a_k$; $V(a_k)$ = set of CSC violations removed by $a_k$; Min-cut = minimal reference-cut set; Min-ref = minimal reference set (transitions in the bracket are included if the delays of transitions $x^*$ and $\overline{x^*}$ cannot be considered separately); + = no permissible arc for $s_6$; *= $a_1(a_3)$ is found for removing the predecessor state of $s_2(s_{10})$

number of candidate arcs for removing a violation is generally more than one. Furthermore, if a predecessor state of the removed target can give delay arcs removing fewer states, these arcs are also candidates. In step 2, we check the covering relation between each arc derived in step 1 and each violation. Although an arc $a_k$ is not derived for some violation, it can also remove this violation. If such, the violation is also put into the set of violations removed by $a_k$, $V(a_k)$. Step 3 then derives the minimal reference set for each candidate arc (the algorithms proposed in [6, 10] which can evaluate the actual delay difference between two concurrent transitions in a timing STG can be modified for our purpose). Then we can check the evaluation order among delay arcs and creat a directed graph, Evaluation_Order_Graph (EOG), whose vertices represent the delay arcs derived in step 1, and an arc exists from vertex $(a_1 \rightarrow a_2)$ to vertex $(b_1 \rightarrow b_2)$ if $a_2$ belongs to the minimal reference set of arc $b_1 \rightarrow b_2$. In the EOG, a vertex subset which induces an acyclic subgraph represents an arc set satisfying the acyclic_requirement. Note that if the selected arcs satisfy the acyclic_requirement, the resultant STG automatically satisfies Step 2 in the liveness_and_environment_requirement. With the EOG and all $V(a_k)$ we can derive solutions. A solution is an arc subset $A'$ of $A$ whose $V(A')$ covers (removes) all violations and their corresponding vertices in the EOG from an acyclic graph. When there is more than one solution in the EOG, a minimal cover is derived in our algorithm. Finally, the selected delay arcs with an evaluation order are returned for later hazard analysis. Note that a solution does not always exist for this algorithm to remove all CSC violations (due to no reference-cut set for some necessary arc or no consistent delay arc set). Our algorithm then reports the subset of CSC violations which can be removed by delay padding.

Let us take the the STG of Fig. 1*a* to illustrate the algorithm. Table 2 shows the derived delay arcs for all CSC violations and the corresponding minimal reference set for each arc. Fig. 5*a* and *b* show the two EOGs for the arc set in Table 2 based on two different design considerations. Let us derive a solution from Fig. 5*b*, which could be either $\{a_1, a_3\}$ or $\{a_3, a_4\}$ since both cover all violations and meet the acyclic_requirement, but not $\{a_4, a_2\}$, since they violate the requirement. Finally, the required evaluation order (i.e. a topological order in the selected subgraph) is determined. In the example of $\{a_1, a_3\}$, both $a_1 \rightarrow a_3$ and $a_3 \rightarrow a_1$ are allowed.

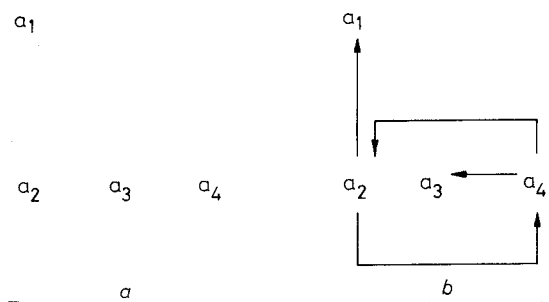


**Fig. 5** *Evaluation_Order_Graph derived assuming that rising and falling transitions can be slowed separately, and the one derived without the assumption*
*a* With assumption; *b* without assumption

The complexity of this algorithm depends on the number of CSC violations $V$ and of candidate arcs $C$. After the preprocessing for all transition relations [9], the first two steps traverse the SG at most $O(V + C)$ times. Step 3 needs to traverse the STG at most $O(C)$ times. However, the constrained minimal covering problem in step 5 is not a polynomial. For a larger set of CSC violations, efficient heuristics still need to be explored.

## 5 Delay padding for removing CSC violations

In this Section we show that a hazard-free circuit can always be derived if a consistent delay-arc set exists for all CSC violations. A procedure will be proposed to satisfy all delay arcs by delay padding and produce a hazard-free circuit. This procedure is modified from the procedure for hazard removal in [8]. Basically, the detection for all possible hazards directly adopts the method in [8], but the hazard removal needs some modification.

All possible causes of CSC violations in the physical circuit can be discovered by using the existing hazard-detection procedure in [8]. A hazardous case is detected in that procedure if the STG-specified order of two transitions could be reversed to cause the cubes of some signal circuits to be turned on and off in the wrong order. Specifically, the disordering causes an unspecified state-sequence to excite an unexpected output (a hazard). For an STG with delay arcs, the disordering may occur between two transitions whose order is specified by a delay arc. Such a disordering recovers a problematic state to affect noninput circuits.

Thus, a CSC violation in the physical circuit appears. Procedure 7.1 of [8] detects all transition pairs whose disorderings cause hazards on some outputs, and checks with some timing inequalities to make sure whether the disorderings do occur (thus hazards do occur) in the physical circuit. Using its detection part for all outputs, we can also identify all those reversed transition pairs, each of which is ordered due to a delay arc. These identified pairs will be the input to our procedure Eliminate_Hazard().

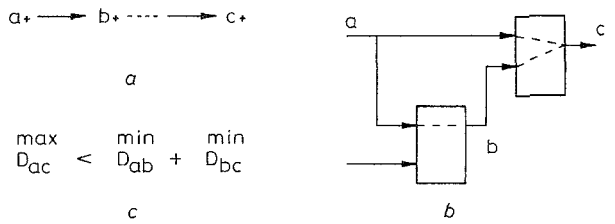


**Fig. 6** *An STG segment, the corresponding circuit and delay inequality*
*a* STG segment; *b* corresponding circuit; *c* delay inequality

However, the analysis of delay inequalities and the delay padding for removing hazards in [8] cannot be directly applied for delay arcs. We firstly recall their technique. Fig. 6*a* shows an STG segment, in which a hazard in circuit $c$ occurs if the order $a+ \rightarrow b+$ is reversed. To ensure the order $a+ \rightarrow b+$ in affecting $c$, an upper bound on the delay from $a$ to $c$, $D_{ac}^{max}$, must be less than a lower bound on the delay from $a$, through $b$, to $c$, $D_{ab}^{min} + D_{bc}^{min}$. If the inequality is not satisfied, delays must be padded in the output terminal of $b$ to slow down $b$. The way to pad delays does not introduce any new hazard since the inequality will not be changed by any subsequent delay padding. The reasons are that (i) if $a$ is slowed down later, it affects both sides of the inequality with the same value such that the inequality is retained, and (ii) if $b$ is slowed, this only enlarges the value on the right-hand side of the inequality. All hazards caused by the disordering of any two transitions can be removed by such rules. These, however, cannot be applied directly to the hazard cases resulting from delay arcs. The reason is that a delay arc $x^* \rightarrow y^*$ does not imply that $x$ will be an input signal to the circuit of signal $y$ to produce transition $y^*$, unlike a regular arc in the STG. Therefore, the calculation of delay difference between $x^*$ and $y^*$ cannot be the same as in [8].

We now present the algorithm Eliminate_Hazards in Fig. 7, which accepts all possible causes to hazards due to CSC violations, and finds out all actual hazard cases and then eliminates them. Before this procedure, we need to use the procedure in [8] to remove all those hazards caused by the disordering of any two transitions ordered in the original STG. This procedure can then ensure that all CSC violations are removed and produce a hazard-free circuit. For each hazard case (an ordered transition pair $x^* \Rightarrow y^*$ due to a delay arc and a signal $c$ which has a hazard if their order is reversed) there are four main steps. First, an upper bound on the delay along the circuit path from the minimal reference-cut set to $x$ is derived. The calculation is completed by a recursive procedure upper_bound(), as shown in Fig. 8. First it ensures that enabling signals arrive at the output $c$ no earlier than all the other signals ordered with them. This allows us to determine an upper bound from those enabling signals. The ensured relation is true if the inequality $(D_{wx}^{max} < D_{wd}^{max} + D_{dx}^{max})$ is satisfied for each enabling transition $d^*$. Delays are padded to satisfy it only if it is not true. It is important to note that the inequality will not be changed by any subsequent padded delays for other hazardous cases. The reasons are that (i) if $w$ increases delays, it affects both sides with the same value and hence does not change the inequality, and (ii) if $d$ increases delays, it enlarges the right-hand side and hence also keeps the inequality unchanged. This inequality is easier to meet than those from regular hazards (not from CSC violations). It is reasonable that this inequality is generally satisfied automatically and thus actually delay padding for this is not frequent. Note also that since the inequality henceforth retains unchanged, we need to perform the check only once for any transition. The second part of the upper_bound() then derives the bound. Since the enabling signals have dominated the bound calculation, the minimal reference-cut set in the STG corresponds to a cut set of all critical circuit paths to $x^*$ in the physical circuit. Therefore, the largest one among those circuit paths derived from the cut set to $x^*$ is a correct upper bound.

**Eliminate_Hazard():**
(Given a live STG $G$ with initial marking $m_0$, a circuit implementation of $G$, a consistent delay-arc set $A$ derived form *Select_Consistent_Arc_Set( )* and the set of transition pairs $T$ each of which is ordered due to a delay arc and will cause hazards if their order is reversed. Let $A = \{a_1, a_2, ..., a_n\}$, where the index denotes their evaluation order.)
Foreach delay arc $a_i$, $i = 1, 2, ..., n$
  Foreach transition pair $(x^* \Rightarrow y^*) \in T$ ordered due to $a_i$
    Foreach noninput $c$ which has a hazard if $x^* \Rightarrow y^*$ is reversed
      Let $R = \{r_{1*}, r_{2*}, ..., r_{m*}\}$ be the minimal reference-cut set of $a_i$;
      **/* Derive an upper bound on the delay for $x^*$ along the circuit path from $R$ to $x$ */**
      Let $D_{rx}^{max} = 0$;
      Foreach $r_{k*}$, $k = 1, 2, ..., m$
        $tem$ = upper_bound($r_{k*}$, $x_*$);
        If ($tem > D_{rx}^{max}$), $D_{rx}^{max} = tem$;
      **/* Derive a lower bound on the delay for $y^*$ along the circuit path from $R$ to $y$ */**
      Let $D_{ry}^{min} = infinite$;
      Foreach $r_{k*}$, $k = 1, 2, ..., m$
        $tem$ = lower_bound($r_{k*}$, $y_*$);
        If ($tem < D_{ry}^{min}$), $D_{ry}^{min} = tem$;
      **/* Check delay inequality and pad delays to satisfy it if it is not met */**
      Let $D_{xc}^{max}$ be an upper bound on the delay along the circuit path from $x$ to $c$;
      Let $D_{yc}^{min}$ be a lower bound on the delay along the circuit path from $y$ to $c$;
      If ($D_{rx}^{max} + D_{xc}^{max} < D_{ry}^{min} + D_{yc}^{min}$), no hazard exists;
      Else pad delay $| D_{rx}^{max} + D_{xc}^{max} - D_{ry}^{min} - D_{yc}^{min} |$ to the output terminal of $y$;
      **/* Consider the first run from the initial marking */**
      Let $P$ be the minimal reference set of $a_i$;
      If $m_0$ enables a transition $\in P$,
        Let $R$ be the set of transitions $\in P$ and enabled in $m_0$;
        Rederive a lower bound on the delay for $y^*$ along the circuit path from $R$ to $y$.
        Recheck delay inequality and pad delays if required.

**Fig. 7** *Algorithm to eliminate all hazards caused by CSC violations in physical circuit*

**Upper_bound($r^*$, $x^*$):**
If ($x^* == r^*$) return (0);
Let $T$ be the set of all enabling transitions of $x^*$ in STG;
**/* Ensure $d^* \in T$ arrive $x$ no earlier than a signal ordered with $d^*$ */**
If the following have never been done for $x^*$,
  Foreach $w^* \notin T$ but $w$ is an input in the circuit to exite $x^*$
    Let $d^* \in T$ and be ordered with $w^*$;
    Let $D_{wx}^{max}$ be an upper bound on the delay for $x^*$ along the circuit path from $w$ to $x$;
    Let $D_{dx}^{max}$ be an upper bound on the delay for $x^*$ along the circuit path from $d$ to $x$;
    Let $D_{wd}^{max}$ be an upper bound on the delay for $d^*$ along the circuit path from $w$ to $d$;
    If ($D_{wx}^{max} > D_{wd}^{max} + D_{dx}^{max}$)
      Pad delay $| D_{wx}^{max} - D_{wd}^{max} - D_{dx}^{max} |$ to the output terminal of $d$;
**/* Derive an upper bound on the delay for $x^*$ along the circuit path from $r$ to $x$ */**
Let $D_{rx}^{max} = 0$;
Foreach $t^* \in T$
  If $t^*$ is concurrent with $r^*$, $tem = -\infty$;
  Else $tem$ = upper_bound($r^*$, $t^*$);
  Let $D_{tx}^{max}$ be an upper bound on the delay for $x^*$ along the circuit path from $t$ to $x$;
  If ($D_{rx}^{max} < tem + D_{tx}^{max}$), $D_{rx}^{max} = tem + D_{tx}^{max}$;
Return ($D_{rx}^{max}$).

**Fig. 8** *Procedure to calculate an upper bound on the delay from $r_k^*$ to $x^*$*

The second step is to derive a lower bound on the delay along the circuit path from the minimal reference-cut set to $y$. The procedure is similar to the

upper_bound(), and thus is not presented here. Actually, the derivation of a lower bound is easier than an upper bound. Only enabling signals are sufficient to determine a lower bound for $y^*$ (other signals cannot decrease the bound), since $y^*$ could occur only if all its enabling signals arrive at $y$. The derived bound may be conservative. After these two bounds are derived, we check a delay inequality to detect whether the disordering of $x^* \Rightarrow y^*$ could occur. Delays will be padded to satisfy the inequality if it is not met. The final step is needed if the initial marking is between the reference-cut set and these two transitions $x^*$ and $y^*$. As such, the timing difference between $x^*$ and $y^*$ in the first run could be incorrect. We recalculate a lower bound to ensure their relation.

The correctness of this overall algorithm is established by the following theorem.

*Theorem 2*: If a consistent delay-arc set exists for all CSC violations, all delay-arc orderings can always be satisfied by delay padding with the procedure Eliminate_Hazards() (see proof in Appendix 8.2).

The worst-case running time of the algorithm can be estimated as follows. Suppose that the STG has $n$ signals and $O(n)$ transitions. The number of hazard cases is at most $O(n^3)$ (all transition pairs cause hazards on all signals). To eliminate a hazard, it requires traversing the STG at most $O(n)$ times to calculate the time separation between two transitions. Thus, in the worst case, it takes $O(n^3)$ complexity to eliminate a hazard. In practice, the number of hazards is far less than $O(n^3)$.

## 6 Results and conclusion

The proposed method has been evaluated with the benchmark in sis (a CAD tool of UC Berkeley). Table 3 shows a comparison between the three methods: signal insertion, arc insertion and delay padding. The results of signal insertion and arc insertion are taken from sis, which are optimal designs derived by expert designers or from the literature. For comparison, we assume that each signal is implemented with a complex gate and each transition has a delay ranging from 1 to 2 units. The performance degradation is evaluated in terms of the maximum delay of the longest cycle in an STG. The experimental result shows that the delay padding does not significantly cause more degradation than the other two methods. In the case of atod.g, it is even the best. As for applicability, the delay padding is able to resolve CSC violation better than arc insertion. The three cases (sendr-done.g, atod.g and subf-ram-write.g) which cannot be resolved by the arc-insertion method due to the noninput constraint have been resolved by the delay padding method. For comparison with the signal-insertion method, the only failed case, due to the noninput constraint, is resolved successfully by the delay padding method. However, the delay padding method fails for three cases. The vbe6a.g involves cyclic independencies such that a consistent delay-arc set does not exist. The other two failed cases, nak-pa.g and master-read.g, are caused by certain states which cannot be removed from the original STG. The typical problem in these two STGS is as follows: $s1 \xrightarrow{a+} s_2 \xrightarrow{a-} s_3 \xrightarrow{b+} s_4 \xrightarrow{b-} s_1$, with $s1$ and $s3$ violating CSC. Since neither one can be removed, there is no solution by delay padding or arc insertion. Note that for the successful cases the signal-insertion method results in increased hardware, unlike the delay-padding method, which may resolve CSC problems with the inherent circuit delays without any extra hardware. Furthermore, the delay-padding method can resolve the cases which defy other methods [2, 3] due to the noninput constraint.

We have proposed a novel alternative to remove CSC violations by exploiting delays in the physical circuit. It circumvents the two main drawbacks in the existing methods: the noninput constraint and the possible waste of overhead. The constraints for padding delays to remove CSC violations have been formulated. An algorithm has been proposed to derive a consistent set of constraints to remove all CSC violations. It has been shown that those constraints can always be satisfied by our proposed hazard-elimination algorithm and a hazard-free circuit can be derived. The result stated above has shown the applicability of our approach. In the future, the approach will be extended to more complex models of STGs and other specifications such as the burst-mode and state-based model.

**Table 3: Experimental results**

| STGs | Signal-insertion | | Arc-insertion | | Delay-padding+ | |
|---|---|---|---|---|---|---|
| | pd | transition# | pd | arc# | pd | delay# |
| sendr-done.g | F* | — | F* | — | 29% | 2 |
| vbe4a.g | 11% | 2 | 20% | 3 | 16% | 4 |
| atod.g | 18% | 2 | F* | — | 10% | 2 |
| nousc.ser.g | 33% | 2 | 33% | 2 | 33% | 2 |
| nousc.g | 0 | 2 | 0% | 2 | 10% | 2 |
| subf-ram-write.g | 25% | 4 | F* | — | 25% | 13 |
| nak-pa.g | — | 4 | F | — | F | — |
| vbe6a.g | — | 4 | F* | — | F | — |
| master-read.g | — | 8 | F | — | F | — |

F = failed; F* = failed due to the non input constraint; pd = ratio of performance degradation (increased delay/new total delay); transition# = number of inserted transitions; arc# = number of inserted arcs; delay# = unit number of padded delays; + = it is possible that no actual delay-padding is needed

## 7 References


1 HAUCK, S.: 'Asynchronous design methodologies: An overview', *Proc. IEEE*, 1995, **83**, (1), pp. 69–93

2 LAVAGNO, L., MOON, C.W., BRAYTON, R.K., and SANGIOVANNI VINCENTELLI, A.: 'An efficient heuristic procedure for solving the state assignment problem for event-based specifications', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1995, **14**, pp. 45–60

3 VANBEKBERGEN, P., CATTHOOR, F., VAN MEERBERGEN, J., and DE MAN, H.: 'Optimized synthesis of asynchronous control circuits from graph-theoretic specifications'. Proceedings of the International Conference on *Computer-aided design*, 1990, pp. 184–187

4 VANBEKBERGEN, P., LIN, B., GOOSSENS, G., and DE MAN, H.: 'A generalized state assignment theory for transformations on STGs'. Proceedings of the International Conference on *Computer-aided design*, 1992, pp. 112–117

5 YKMAN-COUVREUR, CH., VANBEKBERGEN, P., and LIN B.: 'Concurrency reduction transformations on state graphs for asynchronous circuit synthesis'. International Workshop on Logic Synthesis, 1993

6 MYERS, C.J., and MENG, T.H.: 'Synthesis of time asynchronous circuits', *IEEE Trans. VLSI Syst.*, 1993, **1**, pp. 106–119

7 CHU, T.A.: 'Synthesis of self-timed control circuits from graphical specifications'. PhD thesis, MIT, June 1987

8 LAVAGNO, L., KEUTZER, K., and SANGIOVANNI, VINCENTELLI, A.: 'Synthesis of hazard-free asynchronous circuits with bounded wire delays', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1995, **14**, pp. 61–86

9 KISHINEVSKY, M.A., KONDRATYEV, A.Y., and TAUBIN, A.R.: 'Specification and analysis of self-timed circuits', *J. VLSI Signal Process.*, 1994, **7**, pp. 117–135

10 AMON, T., HULGAARD, H., BURNS, S.M., and BORRIELLO, G.: 'An algorithm for exact bounds on the time separation of events in concurrent systems'. Proceedings of International conference on *Computer design*, 1993, pp. 166–173

11 COMMONER, F., and HOLT, A.W.: 'Marked directed graphs', *J. Comput. Syst. Sci.*, 1971, pp. 511–523


## 8 Appendix

### 8.1 Proof of theorem 1

We first show that $G'$ with $m'_0$ has an underlying live and safe net. Since each directed cycle has at least one token, the net is live [11]. To show the safeness, we recall that a live marking is safe iff every arc is in a directed cycle with token count 1 [11]. Since $G$ (the original one) has an underlying live and safe net and no arc is removed from it, each original arc in $G$ is still in a directed cycle with token count 1 such that it does not cause unsafeness to $G'$. In other words, only the added arcs may cause unsafeness. If an added arc $t_i \rightarrow t_j$ can carry more than one token through some firing sequence from $m'_0$, then in $G$, $t_i$ can fire twice while $t_j$ keeps enabled. Such an arc is not permissible in the type-3 concurrent behaviour and will not be selected by our algorithm. In other words, $G'$ does not contain such an arc. Consequently, no arc can carry more than one token after any firing sequence from $m'_0$, and the underlying net of $G'$ is safe.

Furthermore, all the added arcs do not remove any ordering relation between transitions from $G$. Those added arcs, except for type 2, also do not carry any token in $m'_0$, so that the firing orders from the initial marking in $G$ are preserved in $G'$. As a result, $G'$ specifies a subset of all possible transition sequences specified by $G$ from $m_0$. That is, behaviour observed by the environment from $m_0$ remains the same. Consequently, this also ensures that the rising and falling transitions of a signal occur alternatively as $G$. Since the underlying net has already been proved live and safe, $G'$ is therefore live.

### 8.2 Proof of theorem 2

The proof is based on the bounds derived from upper_bound() and from lower_bound(). We compare all those upper bounds from all transitions in $R$ (the minimal reference-cut set) to $x^*$ and then select the largest one as $D_{rx}^{max}$ Similarly, we select the smallest one as $D_{ry}^{min}$ from those lower bounds from all transitions in $R$ to $y^*$. If the inequality $D_{rx}^{max} + D_{xc}^{max} < D_{ry}^{min} + D_{yc}^{min}$ is met, then for each $_k^* \in R$, an upper bound $U(r_k^*)$ on the delay for $x^*$ along the circuit path from $r_k$, through $x$, to $c$, is less than a lower bound $L(r_k^*)$ on the delay for $y^*$ along the circuit path from $r_k$, through $y$, to $c$. This ensures the relation $x^* \Rightarrow y^*$ in affecting output $c$. If the inequality is not satisfied (a hazard exists), then delays are padded to satisfy it. This hazard will never occur if the relation $U(r_k^*) < L(r_k^*)$ for each $r_k^*$ will not be changed by any subsequent padded delays for other hazardous cases. If this is the case, we can also state that the delays padded for $D_{rx}^{max} + D_{xc}^{max} < D_{ry}^{min} + D_{yc}^{min}$ do not introduce new hazards elsewhere. Therefore, the key proof is to show that the relation $U(r_k^*) < L(r_k^*)$ for each $r_k^*$ henceforth remains unchanged.

First we show that the delay inequality ($D_{rx}^{max} + D_{xc}^{max} < D_{ry}^{min} + D_{yc}^{min}$) will not be changed by any subsequent padded delays if no delays are padded to the circuits of $R$. Note that the delay inequality keeping unchanged is sufficient to ensure that $U(r_k^*) < L(r_k^*)$ for each $r_k^*$ is retained. Since those hazards caused by the disordering of any two transitions in the original STG have been removed, we only need to consider the cases of delay arcs. If all the delay arcs are treated according to the evaluation order derived in the algorithm in Fig. 4, the circuit delays on all signals of the minimal reference set for a delay arc will not be modified by any subsequent processes for other hazardous cases. Hence, the left-hand side of the delay inequality will retain unchanged. If delays are increased on $D_{ry}^{min}$ or $D_{yc}^{min}$, this only enlarges the right-hand side. Hence, the inequality is still retained. Furthermore, delays increased on the output $c$ do not affect the inequality. In summary, the inequality will not be changed by any subsequent padded delays for other hazardous cases.

We now consider the case of delays increased on some $r_k^* \in R$. These increased delays actually affect the upper bound from $r_k$ to $x$ and the lower bound from $r_k$ to $y$ both with the same value. Hence, the relation $x^* \Rightarrow y^*$ in affecting output $c$ still holds.