Eliminating Isochronic-Fork Constraints in Quasi-Delay-Insensitive Circuits

Nattha Sretasereekul

RCAST
The University of Tokyo
Tokyo, 153-8904
Tel: +81-3-5452-5167
Fax: +81-3-5452-5161
nattha@hal.rcast.u-tokyo.ac.jp

Takashi Nanya

RCAST
The University of Tokyo
Tokyo, 153-8904
Tel: +81-3-5452-5160
Fax: +81-3-5452-5161
nanya@hal.rcast.u-tokyo.ac.jp

Abstract— The *Quasi-Delay-Insensitive (QDI)* model assumes that all the forks are isochronic. The isochronic-fork assumption requires uniform wire delays and uniform switching thresholds of the gates associated with the forking branches. This paper presents a method for determining such forks that do not have to satisfy the isochronic fork requirements, and presents experimental results that show many isochronic forks assumed for existing QDI circuits do not actually have to be "isochronic"or can be even ignored.

I. Introduction

Delay-insensitive (DI) circuits are a class of asynchronous circuits whose correct operation is independent of any delays in gates and wires. The delay insensitivity is attractive, especially with deep-submicron technologies in which uncertainty in process parameters may increase.

In practice, however, the class of completely DI circuits is extremely limited [1]. Then as a compromise, the isochronic fork assumption is introduced. DI circuits with the isochronic-fork assumption are known as *quasi-delay-insensitive (QDI)* circuits.

The isochronic-fork assumption requires uniform wire delays and uniform thresholds of gates in VLSI fabrication [1, 2]. As a matter of fact, several methods to make sure the validity of the isochronic-fork assumption have been proposed [2, 3]. However, careful examination of possible transition races caused by different branches of an isochronic fork leads us to an observation that some of the isochronic forks do not have to be satisfied.

This paper presents a method for determining such isochronic forks, and presents experimental results that show many isochronic forks assumed for existing QDI circuits do not actually have to be "isochronic" or can be even ignored.

II. QDI circuits

A digital circuit is called the delay-insensitive circuit if its correct operation does not depend on delays in circuit elements, i.e., gates and wires. This behavior strictly relies on the *acknowledgment requirement* that every transition on each circuit variable must be acknowledged by causing a transition on some other circuit variable.

Since DI circuits can be built from only C-elements, single input gates (buffers and inverters) and wires, only a small class of circuits is allowed [1]. In practice, combinatorial gates (such as AND, OR, etc.) and forks (or branched wires) are necessary to construct any useful circuits. The branched wires which are connected to the combinatorial gates always cause unacknowledged transitions that make the circuit lose delay-insensitivity, and may cause unexpected hazards.

Quasi-delay-insensitive (QDI) circuits are delay-insensitive under the isochronic-fork assumption. The isochronic fork has been defined in terms of acknowledgment by Martin [1] as follows.

"In an isochronic fork, when a transition on one output is acknowledged, and thus completed, the transitions on all outputs are acknowledged, and thus completed."

The introduction of the isochronic-fork assumption to the DI model is sufficient to construct any useful circuits.

III. Don't-worry Isochronic Forks

By its definition, the implementation of an isochronic fork requires that all the branches have the same wire delay and all the receiving gates have the same switching threshold. These requirements may be difficult to be satisfied in practice. However, when we carefully examine isochronic forks and properties of the gates driven by these forks, we see that some of the claimed isochronic forks do not actually have to be isochronic since they do not introduce any hazard to the circuit. Such isochronic forks are called *don't-worry isochronic forks*, for which we

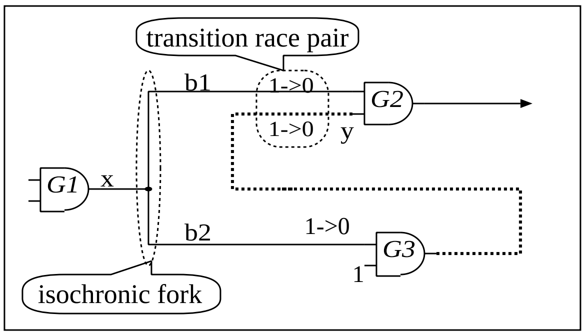


Fig. 1. An isochronic fork and a transition race pair.

do not have to worry about any delay constraints for their implementation.

Since unacknowledged transitions may cause hazards, we examine all possible transition races caused by these transitions. Fig.1 shows an isochronic fork with branches $b1$ and $b2$. Suppose the down-going transition of $b1$ (expressed as $b1-$) is unacknowledged. Since $b1-$, without isochronic-fork constraints, is not guaranteed to occur at the expected time, the transition races between $b1-$ and other input transitions of $G2$ must be examined. That is the transition races between $b1-$ and $y-$, and between $b1-$ and $y+$ are examined whether they cause hazards or not. For the ease of later referring, we call each transition race a *transition race pair.*

We can observe that if the gate $G2$ in Fig.1 is the n-input gate, the unacknowledged transition $b1-$ causes $2(n-1)$ transition race pairs to be checked. Since we analyze, at a time, one unacknowledged transition such as $b1-$, it is sufficient to consider only pairs of transitions formed by $b1-$.

A transition race pair is called a *do-worry* transition race pair if it can possibly cause hazards, and is called a *don't-worry* transition race pair if it can cause no hazard. There are three conditions for a transition race pair to be hazard-free, thus don't-worry.

Condition 1: *The transition race does not constitute any multiple input change for a function hazard.*

We know that a transition race of inputs of a 2-input AND gate is always hazard-free when the transition race is 00 → 11 or 11 → 00, but may possibly cause static-0-hazard when the transition race is 01 → 10 or 10 → 01. In Fig.1, if the transition race of inputs of $G2$ is always 00 → 11 or 11 → 00, this isochronic fork becomes don't-worry. In contrast, if the transition race can be 01 → 10 or 10 → 01, this isochronic fork is do-worry. Similar arguments hold for general case of n-input gates and n>2.

Condition 2: *The transition race occurs when the gate is disabled.*

In addition to condition 1, in case of n-input gates, any transition race pairs can not cause any hazards when the gates are disabled.

The last condition needs the single assumption below:

A single wire delay is less than any delay of signal propagation path that passes through the environment outside the circuit.

In Fig.1, when transition race pair $(b1-, y+)$ occurs, the transition race can be hazard-free under the constraint that the delay of the path from $x-$ to $b1-$ is less than the delay of the path from $x-$ to $y+$. Therefore, a transition race pair is considered to be hazard-free if its constraint satisfies the following condition.

Condition 3: *If path $d1$ is a wire inside the circuit while path $d2$ passes through the environment outside the circuit, then the constraint that requires (delay of $d1$) $\leq$ (delay of $d2$) is already satisfied.*

IV. Algorithms

Given a logic gate net-list and its behavior specification, we search the forks, determine whether these forks have to be isochronic or not, examine whether the isochronic forks are do-worry or don't-worry, and derive layout constraints for implementing the do-worry isochronic forks to ensure DI correct operations.

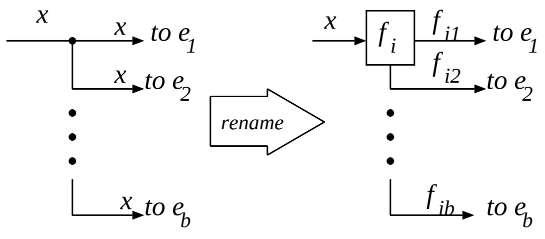
In this paper, the logic gate net-list in BLIF [4] format and its behavior specification in STG format are used.

A. Determining forks

Since a circuit under consideration must be closed, a given gate logic net-list (BLIF) is transformed into the closed circuit by representing the environment of the circuit as gates. From the closed circuit, the forks can be determined as follows.

Assume a closed circuit net-list S=(V,E), where V is a set of circuit variables v_1, v_2, ..., v_n and E is a set of circuit elements e_1, e_2, ..., e_m. Each variable is the output of one element, but may be the input of one element or more elements. The variable that is the input of more than one element introduces a fork. We define fork elements as follows.

Definition *Let x be a variable of a closed circuit. If x is the input of elements e_1, e_2, ..., e_b, where b≥2, then x is said to be the input of the* **fork** *f_i which has renamed branched wires f_{i1}, f_{i2}, ...,f_{ib} as outputs.*



```
Procedure DFORKS( $V$,$E$)
  Inputs: A logic net-list $S = (V, E)$ in BLIF format.
  Outputs: A logic net-list $S' = (V', E')$ including forks
    as circuit elements in BLIF format.
  Operations: Find forks, add forks as gates to BLIF,
    and rename branched wires of forks.

1   foreach variable $v_i \in V$ do
2       $b = 0$
3       foreach element $e_j \in E$ do
4           if $v_i \in \{inputs\ of\ e_j\}$ then
5               $b = b +$ the number of inputs of $e_j$ that
                                      are fed by $v_i$
            end if
        end foreach
6       if $b \geq 2$ then
7           $create\_fork(v_i, b)$
8           $rename(v_i)$
        end if
    end foreach
```

Fig. 2. Algorithm for determining forks in a closed circuit.

A fork element will be considered as a gate, but it is different from the normal gates in the sense that it has multiple outputs while the normal gates have only one output. A k-input gate can be expressed as $[i_1, i_2, ..., i_k ; o_1]$, while a b-output fork can be expressed as $[i_1 ; o_1, o_2, ..., o_b]$. We use ";" to separate inputs and outputs of the gate.

Procedure DFORKS of Fig.2 shows the algorithm for determining forks in a closed circuit. The output of the procedure will be the new logic net-list S'=(V', E') that includes forks in BLIF format.

For each variable $v_i \in V$, an auxiliary b is used for counting the number of possible branches of v_i. A variable v_i may be fed to one or more inputs of an element e_j. If $b \geq 2$, the fork element with input v_i is created by the function *create_fork()*, and the name of v_i at the corresponding elements is changed by the function *rename()*.

This algorithm performs in $O(n^2)$, where n is the number of circuits variables. After applying this algorithm, the number of variables increases to be n' because branch variables of forks and fork elements are added. n' is equal to the number of *fan-ins* of gates in the net-list.

B. Determining isochronic forks

A fork that contains at least one unacknowledged branch transition is an isochronic fork. In this paper, we introduce the *extended STG (E-STG)* to determine isochronic forks. The E-STG is the STG that has been added the consideration of forks. We can easily identify the unacknowledged branch transitions from the E-STG.

Procedure DISOF of Fig.3 shows the algorithm for determining isochronic forks. Inputs for the procedure are the logic net-list created by procedure DFORKS, STG, and gate logic function and properties. Outputs of the procedure are an E-STG, a sequence of circuit states and a set of unacknowledged transitions.

```
Procedure DISOF($V'$, $E'$, $init$, $start$)
  Inputs: A logic net-list (created by DFORKS)
          $S' = (V', E')$ in BLIF format,
          STG,
          gate logic function and properties.
  Outputs: An extended STG $G = (G_S, G_D)$,
           a sequence of circuit states $B$,
            a set of unacknowledged transitions $U$.
  Operations: Create a sequence of circuit states,
              create E-STG, and then
              unacknowledged transitions are
              informed by E-STG.

1    $U = \{\ \}$
2    foreach variable $v_i \in V'$ do
3        $EQ[i] = create\_equation(v_i)$
     end foreach
4    $P = N = init$
5    $N = start$
6    $B = B \bigcup N$
7    while $(N \neq P)$ and $(N \neq init)$ do
8        $P = N$
9        $N = compute\_new\_state(EQ)$
10       $B = B \bigcup N$
11       for $(i = 1, 2, . . ., n')$ do
12           if $N[i] \neq P[i]$ then
13               $T = transitions(N[i], P)$
14               $G = G \bigcup T$
             end if
         end for
     end while
15   foreach $d \in G_D$ do
16       if $d \notin G_S$ then
17           $U = U \bigcup \{d\}$
         end if
     end for
```

Fig. 3. Algorithm for determining isochronic forks.

In the DISOF procedure, *start* represents the start conditions. The start conditions of a circuit can be determined from the initial marking of the given STG. For example, for a circuit with initial marking "*.marking* { $A+$, $B+$}", there are two start conditions: starting with $A+$ and starting with $B+$. We analyze the circuit with each start condition separately, and then combine all results together at the last step.

An equation set EQ is an auxiliary array that is used for computing the next value of V'. $EQ[i]$ is the output logic function of the element whose output is v_i. The value of $EQ[i]$ is computed from the element input variable values in the present state.

Each variable $v_i \in V'$ of the net-list has its own equation that is created by the function *create_equation()* (line 3).

An auxiliary array P is used for keeping the value of each variable $v_i \in V'$ in the present state. An auxiliary array N is used for keeping the value of each variable $v_i \in V'$ in the next state. Each element of P and N are initially settled to have the same value with each element of *init* which is the array that keeps the initial values of the circuit (line 4). Then the element of N that corresponds to *start* is settled to have the start value (line 5).

For each time we analyze the circuit with each start condition, a set B keeps a sequence of circuit states (events). Note each B is not the full set of the reachable states of the circuit.

An array $G = (G_S, G_D)$ is a set of parts of E-STG. Each part of E-STG is in [*source transition* , *destination transition*] format. G_D is a set of destination transitions that corresponds to G_S that is a set of source transitions.

An auxiliary array T is also in the same format as G. It is used for keeping the results from the function $transitions(N[i], P)$ which computes the transitions that cause the transition of $N[i]$ (line 13). Then it is added to G (line 14). The transitions that cause the transition of $N[i]$ can be determined by checking the value of the each input of $EQ[i]$. This depends on the properties of individual gates.

The complete B and G are performed by *while loop* of the procedure. The conditions to stop the repetition are when the circuit gets to the steady state ($N = P$) or when the circuit gets back to the initial state ($N = init$).

While the repetition do not get to the stop condition, the next state N is computed by the function $compute\_new\_state(EQ)$ (line 9), N is added to B (line 10), T is computed if there exist $N[i] \neq P[i]$ (line 13), and T is added to G (line 14).

U is a set of unacknowledged transitions. An unacknowledged transition can be determined by checking the sets G_D and G_S. Once a transition occurs ($d \in G_D$), but never causes any other transition ($d \notin G_S$) , then this transition is determined to be unacknowledged. A set of isochronic forks is automatically informed by a set of unacknowledged transitions.

This algorithm performs in $O((n')^2)$. The worst case is of line 16 (checking each destination transition with source transitions).

C. Determining don't-worry isochronic forks

The last step is the determination that which isochronic fork must be satisfied, i.e. the do-worry isochronic fork, and which isochronic fork do not have to be satisfied, i.e. the don't-worry Isochronic fork. The conditions described in section (III) are used in this determination.

Procedure DCNST of Fig.4 shows the algorithm for determining don't-worry and do-worry isochronic forks, and creating layout constraints for the do-worry ones. Inputs for the procedure are a logic net-list $S' = (V', E')$, an extended STG $G = (G_S, G_D)$, a sequence of circuit states B, a set of unacknowledged transitions U, and gate hazard properties. The output of the procedure is a set of do-worry isochronic forks with layout constraints.

```
Procedure DCNST($V', E', G, B, U$)
  Inputs: A logic net-list (created by DFORKS)
          $S' = (V', E')$ in BLIF format,
          an extended STG $G = (G_S, G_D)$,
          a sequence of circuit states $B$,
          a set of unacknowledged transitions $U$,
          gate hazard properties.
  Outputs: Layout constraints for worried
           isochronic forks.
  Operations: Select hazardous gates,
              create transition race pairs,
              examine each race pair is hazardous
              or hazard-free, and
              create layout constraints for
              the hazardous transition race pairs.

1     $RPAIR = \{ \}$
2     foreach $e \in E'$ do
3         if $\exists u \in U$ such that $u.name \in V'$
                  and $u.name \in \{ inputs\ of\ e \}$ then
4             $R = create\_race\_pair(u, e)$
5             $RPAIR = RPAIR \bigcup R$
          end if
      end foreach
6     foreach $r \in RPAIR$ do
7         if $ck\_hzfree(r, B) = true$ then
8             $RPAIR = RPAIR - \{r\}$
9         elseif $ck\_gdisable(r, B) = true$ then
10            $RPAIR = RPAIR - \{r\}$
11        elseif $ck\_extpath(r, E', G) = true$ then
12            $RPAIR = RPAIR - \{r\}$
          else
13            $create\_constraint(r, G)$
      end foreach
```

Fig. 4. Algorithm for determining don't-worry and do-worry isochronic forks, and creating layout constraints for the do-worry ones.

An auxiliary array $RPAIR$ is a set of all possible transition race pairs that are caused by all unacknowledged transitions. An auxiliary array R is a set of possible transition race pairs that are caused by unacknowledged transitions of inputs of each gate $e \in E'$.

An auxiliary variable u represents an unacknowledged transition consisting of *u.name* which is the name of $v_i \in V'$, and *u.tran* which is the transition of *u.name*. If there exist an input of the gate $e \in E'$ that is an *u.name* of $u \in U$, the transition race pairs between u and other input transitions of the gate is then created by the function $create\_race\_pair(u, e)$ that sends the result to R (line 4). Then R is added to $RPAIR$ (line 5).

Next, each transition race pair is checked whether it satisfies conditions 1, 2 and 3 or not. If the transition race pair satisfies at least one of these three conditions, we say that it is the hazard-free transition race pair, and

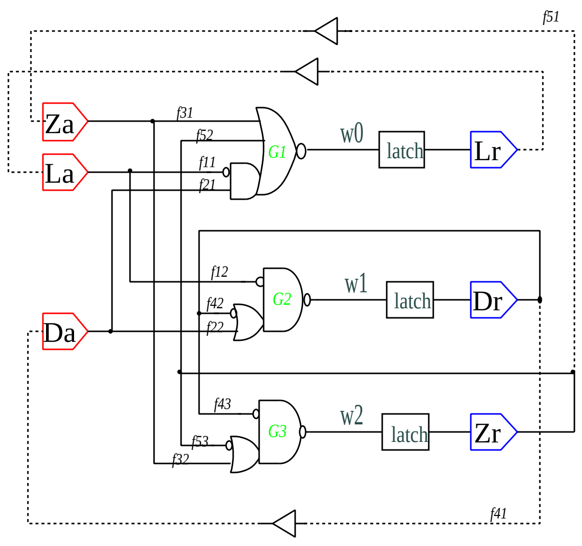


Fig. 5. The circuit diagram implemented with CB-C10 library [5].

then remove it from the set $RPAIR$.

Function $ck\_hzfree(r, B)$ considers each transition race pair $r \in RPAIR$ by using gate hazard properties and the sequence of events B. This function checks on whether the transition race pair is satisfying the condition 1 or not.

Function $ck\_gdisable(r, B)$ also considers each transition race pair $r \in RPAIR$ by using gate hazard properties and the sequence of events B. It checks on whether the gate is satisfying the condition 2 or not.

Function $ck\_extpath(r, E', G)$ checks on whether the paths from the origin of the fork to the gate under consideration are satisfying the condition 3 or not.

Finally, if the transition race pair does not satisfy any conditions, it must be carefully implemented under the constraint that one transition must occurs before the other transition. The constraint is derived by the function $create\_constraint(r, G)$.

This algorithm performs in $O((n')^2)$. The worst case is of line 7 (checking each transition race pair).

D. An Example

A practical example is shown here. By algorithm DFORKS, we obtain the new net-list of a controller as shown in the diagram of Fig.5. The dot lines represent environment paths. The new name of each branch of forks is defined.

From this circuit diagram, their E-STG that includes the consideration of forks are obtained by algorithm DISOF as shown in Fig.6. In each fork, a branch transition that is not followed by any transition is unacknowledged. Such a branch transition is identified by a dot circle. The fork that contains at least one unacknowledged branch transition is the isochronic fork.

Finally, by algorithm DCNST, we obtain the do-worry constraints. The result of the analysis is shown in Fig.7.

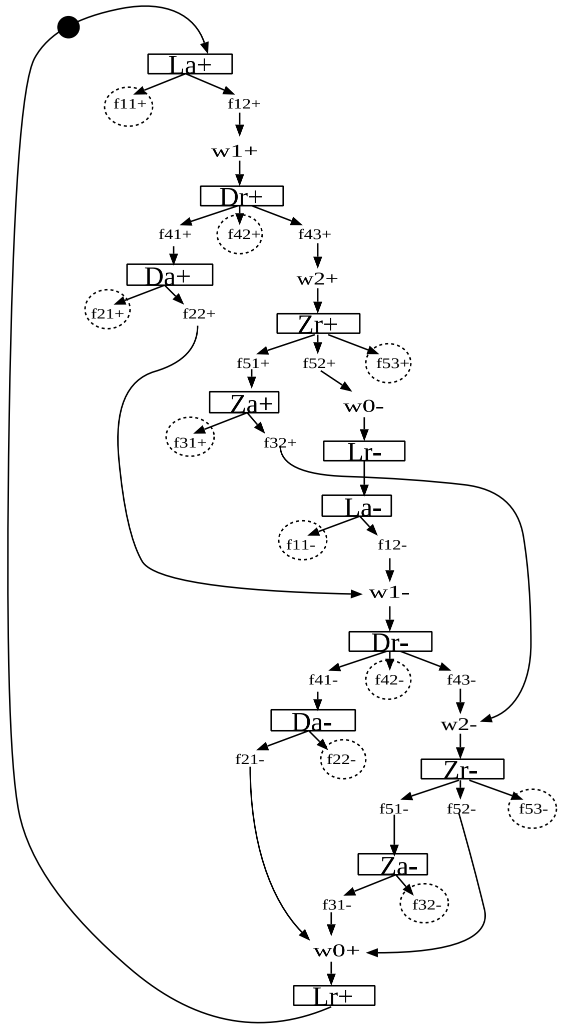


Fig. 6. The extended STG (E-STG).

We can observe that there exist many unacknowledged transitions as shown in Fig.6. Among transition race pairs caused by these unacknowledged transitions, only three transition race pairs cause do-worry constraints.

To illustrate the elimination, the analysis of fork $f2(Da; f21, f22)$ of Fig.5 is described as follows. In Fig.6, since transitions $f21+$ and $f22-$ are unacknowledged, the fork $f2(Da; f21, f22)$ is isochronic. All transition race pairs caused by these transitions are examined. The transition race pairs caused by *f21+* are $(f21+, f11+)$, $(f21+, f11-)$, $(f21+, f31+)$, $(f21+, f31-)$, $(f21+, f52+)$ and $(f21+, f52-)$. The pairs $(f21+, f11-)$, $(f21+, f31+)$ and $(f21+, f52+)$ satisfy condition 1, while the pairs $(f21+, f11+)$, $(f21+, f31-)$ and $(f21+, f52-)$ satisfy condition 3. That is there is no hazardous transition race caused by the unacknowledged transition $f21+$. With the similar consideration, there is also no hazardous transition race caused by the unacknowledged transition $f22-$. Clearly, the isochronic fork $f2(Da; f21, f22)$ is the don't-worry isochronic fork.

```
# The total of transition race pairs  = 48
# Do-worry constraints are listed below:
(1) transition race pair (f11+,f52+):
Delay(La+ -> f11+) must be less than Delay(La+ -> f52+).
Delay(La+ -> f52+) is
Delay(La+ ->f12+ ->w1+ ->Dr+ ->f43+ ->w2+ ->Zr+ ->f52+)
------------------------------
(2) transition race pair (f31+,f52-):
Delay(Za+ -> f31+) must be less than Delay(Za+ -> f52-).
Delay(Za+ -> f52-) is
Delay(Za+ ->f32+ ->w2- ->Zr- ->f52-)
------------------------------
(3) transition race pair (f11-,f52-):
Delay(La- -> f11-) must be less than Delay(La- -> f52-).
Delay(La- -> f52-) is
Delay(La- ->f12- ->w1- ->Dr- ->f43- ->w2- ->Zr- ->f52-)
------------------------------
```

Fig. 7. The result of isochronic fork analysis.

TABLE I
Experimental results.

circuit	forks(branches)	race pairs	do-worry race pairs
chu133	6(15)	52	3
alloc_outbound	7(18)	54	3
nak_pa	5(18)	68	8
vbe5b	5(13)	54	2
vbe5c	5(12)	48	3
nowick	6(19)	76	0
converta	6(22)	148	12
half	3(7)	14	2

V. Experimental results

The algorithms for eliminating isochronic-fork constraints presented in the previous section have been implemented. The results obtained from a set of well-known benchmarks (STGs) are shown in Table I. All circuits are implemented with CB-C10 library [5] by using Petrify [6]. (Note that the Petrify tool generates speed-independent (SI) circuits that are equivalent to QDI circuits.)

In Table I, the first column show the name of benchmark circuits, the second column shows the number of isochronic forks and branches (in the parenthesis) and the third column shows the number of transition race pairs caused by unacknowledged branch transitions. The fourth column shows the number of do-worry transition race pairs which are the results of the proposed algorithms having been applied to eliminate don't-worry race pairs.

From Table I, we see that only a small number of do-worry transition race pairs remain. In addition, some do-worry transition race pairs are very easy to be satisfied and therefore, may be ignored. For example, in Fig.7, although the transition race pair $(f11+, f52+)$ is determined to be a do-worry race pair, the constraints required can be considered to be already satisfied, because the delay of the path $(La+ \rightarrow f52+)$ is much larger than the delay of the path $(La+ \rightarrow f11+)$.

VI. Conclusions

We have shown a method to determine which isochronic-fork constraint does not have to be satisfied while preserving the DI correct operation. We also presented algorithms to analyze isochronic forks by introducing extended state transition graph (E-STG).

The algorithms have been implemented, and some benchmarks are tried. From the experimental results, we have shown that many of the isochronic-fork constraints claimed in QDI design can actually be neglected in the layout or fabrication processes.

Our algorithms can be used for evaluating the safety level of the technology mapping tools to obtain logic circuits.

Acknowledgments

This work was supported in part by the Semiconductor Technology Academic Research Center(STARC).

References

[1] Alain J. Martin, "The Limitations to Delay Insensitivity in Asynchronous Circuits", 6th MIT Conference on Advanced Research in VLSI Processes, pp. 263-277, 1990.

[2] Kees van Berkel, "Beware the Isochronic Fork", INTEGRATION, The VLSI Journal 13, pp.103-128, 1992.

[3] Kees van Berkel, "Stretching Quasi Delay Insensitivity By Means of Extended Isochronic Forks", Second Working Conference on Asynchronous Design Methodologies, 1995, pp. 99-106.

[4] University of California Berkeley, "Berkeley Logic Interchange Format (BLIF)", http://kalliope.uni-trier.de/~wagner/html/bdd_html/docu/blif/index.html

[5] NEC Corporation, "CB-C10 Family 0.25-μm CMOS Cell-Based IC (2.5V) Block Library", July 1999.

[6] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers", IEICE Transactions on Information and Systems, Vol. E80-D, Number 3, pp. 315-325, March 1997.