Redressing Timing Issues for Speed-Independent Circuits in Deep Submicron Age

Yu Li, Terrence Mak and Alex Yakovlev
School of EECE, Newcastle University
NE1 7RU, United Kingdom
{*yu.li1, terrence.mak, alex.yakovlev*}*@ncl.ac.uk*

***Abstract*— The class of speed independent (SI) circuits opens a promising way towards tolerating process variations. However, the fundamental assumption of speed independent circuit is that forks in some wires (usually, large percentage of wires) in such circuits are isochronic; this assumption is more and more challenged by the shrinking technology. This paper suggests a method to generate the weakest timing constraints for a SI circuit to work correctly under bounded delays in wires. The method works for all SI circuits and the generated timing constraints are significantly weaker than those suggested in the current literature claiming the weakest formally proved conditions.**

I. Introduction

As technology shrinks into the deep sub-micron scale, process variations become one of the main obstacles to circuit design. Asynchronous design, which inherently highly tolerates process variations, suggests a promising solution to combat this problem. Among all asynchronous design paradigms, delay insensitive (DI) circuits show the highest tolerance to delay variations in both logic gates and wires. However, as was proved in [1], almost all useful asynchronous controller specifications do not have DI implementations. So, speed independent/quasi-delay insensitive (SI/QDI) circuits which only need an isochronic-fork timing assumption, were introduced to enlarge the class of specifications that could be synthesized while still holding a strong variation tolerance ability.

However, the isochronic fork assumption becomes unreliable during process shrinking. The more and more severe threshold variations and wire delays (compared to gate delays), together with buffer insertion, could cause isochronic fork failure. Recent research proved that the isochronic-fork timing assumption could be relaxed into a weaker and easier to satisfy timing assumption [3], and only a few forks in SI circuits are dangerous [2]. These studies provide valuable resources on the hazard detection when isochronic fork assumption is relaxed. But the problem is still there. In [3] authors proved that *adversary path timing assumption* was equivalent to the isochronic fork timing assumption for the correctness of SI circuits. They claimed this assumption was the weakest timing assumption that was both necessary and sufficient for correct operation of SI circuits. However, the isochronic fork timing assumption is only a sufficient but not a necessary condition for the correctness of SI circuits. Thus, the equivalence between adversary path and isochronic fork timing assumptions cannot prove the necessity of the adversary path timing assumption. By considering the function of gates, the timing requirement for correctness is considerably weaker than simply 'contains no adversaries'. In [2] authors proposed a method to generate the timing constraints for a SI circuit to work correctly, but their technique directly compared transitions in the high level signal transition graph to decide whether a circuit glitches. So, the use of their method is quite restricted. In this paper, a technique to generate the weakest timing constraints for a SI circuit to work correctly corresponding to a given environment under the bounded wire delay model is presented. All the generated constraints could be fulfilled by delay padding. Thus, severe process variations in deep submicron age could be tolerated by SI circuits in which a few dangerous forks are fixed.

This paper is organized as follows: section II introduces the concepts used the following sections; section III presents our method to generate the weakest timing constraints for any SI circuit and the how to fulfill them; section IV shows the experiment results and section V concludes the paper.

978-3-9810801-7-9/DATE11/©2011 EDAA

II. Preliminaries

In this section, definitions and notation used in this paper are introduced.

All the wires and gates are considered under pure delay model. Pure delays only shift transitions for a given time without absorbing any glitches; while an inertial delay not only delays a transition but also suppresses any pulses whose width is narrower than a given value. Without considering the case where glitches are used to eliminate the hazards [4], a SI/QDI circuit will work correctly under the inertial delay model if it works correctly under the pure delay model.

Usually, circuits are considered in context with their environment (*ENV*). The signals that come from the *EVN* are inputs to the circuit and denoted by set I and that feedback to the *ENV* are outputs denoted by set O. Besides, the signals inside the circuit are denoted by set R.

Circuit: A circuit is a triple $C = (A, F, \psi)$, where A is a set of signals, F is a set of functions, such that for each signal $a \in (R \cup O)$, there is a function $f_a \in F$ which computes a and ψ is a labeling function which labels a wire between a

and each fanout signal of a. Given a state s of the circuit (which is a Boolean vector), $f_{a\uparrow}(s) = TRUE$ iff the function f_a evaluates to '1' in s, $f_{a\downarrow}(s) = TRUE$ iff f_a evaluates to '0' in s. In this paper the circuit is under the *intra-operator fork timing assumption* [3], which assumes that in a fork, wires fed to same gate are considered to have the same delay. The behavior of an SI circuit is often depicted by a formal version of a timing diagram, called a signal transition graph, which is a interpreted Petri Net [10].

Petri Net (PN): A Petri Net is a quadruple $N = (P,T,F,m_0)$, where P is a finite set of places, T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, and m_0 is the initial marking. A place p (transition t) is an input place (transition) of a transition t (place p) if $p \times t \in F$ ($t \times p \in F$) and is an output place (transition) of a transition t (place p) if $t \times p \in F$ ($p \times t \in F$). The set of input places (transitions) of a transition t (place p) is denoted by ${}^\bullet t$ (${}^\bullet p$) and the set of output places (transitions) of a transition t (place p) is denoted by $t^\bullet$ ($p^\bullet$). A Petri Net is *safe* if each place could have at most one token at any time. A Petri Net has choice if there exists a place p which has more than one output transitions. A Petri Net is a *free-choice net* if any place p has more than one output transition, then p must be the only input place of all these transition. A Petri Net is a *Marked Graph* (*MG*) if it does not have choice. A safe and free-choice Petri Net could be always decomposed into a set of *MG*s by exhaustively choosing every output transition at each choice place. A transition t_1 (t_2) is a *predecessor transition* (*successor transition*) of transition t_2 (t_1) if $t_1^\bullet \cap {}^\bullet t_2 \neq \oslash$ and denoted by $t_1 \Rightarrow t_2$; $t_1 \overset{\sigma}{\Rightarrow} t_n$ if these is a not NULL sequence $t_1 \Rightarrow t_2 \Rightarrow t_3 \Rightarrow \cdots \Rightarrow t_n$. The set of predecessor transitions (successor transition) of transition t is denoted by ${}^<t(t^>)$.

Signal Transition Graph (STG)[10]: A signal transition graph is a triple $G = (N,A,\lambda)$, where N is the underlying Petri Net, A is a finite set of signals, and λ is a labeling function which labels each transition to $A \times \{+,-\}$. $\forall a \in A$, $a+$ depicts a rising transition on signal a, $a-$ depicts a falling transition on signal a and $a*$ is used to depict either $a+$ or $a-$. The places which only have one pre- and post-transition in a STG are often omitted for brevity. In the following sections, a STG is decomposed into a set of MGs and in each MG segment, all places are omitted and the flow relation $\Rightarrow$ between two transitions is called an *arc*.

The STG in which $A = I \cup O$ that only depict the interactions between the circuit and the environment is called a *specification STG*, denoted by STG_{spec}; while the STG in which $A = I \cup O \cup R$ that depict the whole event order in a circuit is called a *implementation STG*, denoted by STG_{imp}. The method presented in this paper requires that the original STG_{imp} could be decomposed into a set of MGs, in which events have explict orderings. Currently the limitation for the original STG_{imp} are *safe* and *free-choice*. The technique to decompose any *safe* STG into MGs will be left for future work.

STG is usually used as a succinct high level event-based model to represent the orderings of the signal transitions in a circuit. The explicit causality and concurrency between transitions in STG is good for manipulating the relations between transitions, while the verification could be done much easier in a low level state-based model, the *state graph*, which explicitly shows every state a circuit could reach. The state graph corresponding to a STG could be derived from this STG by traversing all markings the STG could reach from m_0 [8].

State Graph (SG)[10]: A State Graph is a quadruple $SG = (A,S,E,\pi)$, where A is a finite set of signals, S is a finite set of states, $E \subseteq S \times S$ is a set of transitions, and π is a labeling function which labels each state with a bit-vector over A. The value of signal a in state s is denoted by *s(a)* and a transition from state s to state s' by firing $a*$ is denoted by $s \xrightarrow{a*} s'$ and $s \xrightarrow{\sigma} s'$ if there is a sequence $s \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'$. Event $a*$ is excited in state s if $s \xrightarrow{a*}$. The positive/negative excitation region of signal a, denoted by ER(a+)/ER(a-) is the set of states in which a+/a- are enabled. The positive/negative quiescent region of signal a, denoted by QR(a+)/QR(a-) is the set of states in which *s(a)*=1/*s(a)*=0 and a is not enabled. If an STG contains multiple instances of the same event, i.e. the different transitions with the same label, then the sub-indices are used to distinguish different regions of the same event, $\text{ER}_i(a+)/\text{ER}_i(a-)$ is the set of maximum connected states in which a+/a- are enabled and correspond to the i-th occurrence of a+/a- in the STG. $\text{QR}_i(a+)/\text{QR}_i(a-)$ is the set of maximum connected states in which *s(a)*=1/*s(a)*=0, a is not enabled and following $\text{ER}_i(a+)/\text{ER}_i(a-)$.

III. Resynthesis and hazard removal method

In this section, a technique to generate the weakest timing constraints for a SI circuit to work under the bounded wire delay model is presented. This technique is based on STG relaxation, which removes one timing ordering in an STG and checks whether a glitch could appear in the new STG. If no the STG will be updated, and if yes a timing constraint will be generated. This process iterates until all timing orderings in the final STG are guaranteed by gate logic or interface protocol or timing constraints. The 'relaxation cases' in this section make sure all the situations under bounded wire delay model are considered, the 'relaxation ordering' ensures the generated timing constraints are the weakest under certain criterion and the 'delay padding' method guarantees all of these timing constraints could be fulfilled. The flow graph of the technique is presented in Fig. 1.

A. Generation the weakest timing assumption

1) deriving local STG: A SI circuit is hazard free if each gate is hazard free under hazard free inputs. This property implies that hazard analysis could be carried out in the local environment for each individual gate to avoid the full state exploration problem. The local environment for a gate a is the STG (or a set of MGs that are equivalent to this STG) that only depicts the transition relations between a and the inputs

Fig. 1. The flow graph of design SI circuit in deep sub-micron age

of a. This could be done by projecting each MG segment of the STG_{imp} on $a \cup fanin(a)$. The projection of a Marked Graph on a subset of signals is achieved by projection of the SG of this MG on these signals and then using the technique for deriving free choice PN from finite transition system in [7] to get the new MG. The projection a SG on a subset of signals is defined as follows.

Projection a State Graph on a subset of signals [6]: Projection a $SG = (A, S, E, \pi)$ on $X \subseteq A$ denoted by $proj_X(SG)$ is another $SG' = (X, S', E', \pi')$, where $\forall s = (a_1, a_2, \cdots, a_n) \in S$; $proj_X(s) = (a_i, a_j, \cdots, a_k)$ that $a_i, a_j, \cdots, a_k \in X$; $proj_X(S) = \{s' | \exists s \in S : proj_X(s) = s'\}$; $proj_X(E) = \{proj_X(s) \xrightarrow{e} proj_X(s') | s \xrightarrow{\sigma} s'$ *and* $\forall e \in X$, *only one is different between* s *and* $s'\}$

The step that derives the local STG for gate_a is performed by function Deriving_local_STG(MG, a).

2) Timing ordering relaxation: In order to allow a SI to work under bounded wire delays, certain timing conditions have to be satisfied on the inputs of the logic gate in the circuit. For this, we need to analyse the timing relations between signal transitions in the local STG(s) for each gate. When we remove the isochronic fork assumption, we effectively relax some of these relations. In particular, the relations that need to be relaxed are between input transtions to the gate. Below is a classification of the four kinds of arcs that appear in the local STG of a gate a.

(1) $x* \Rightarrow y*$, where $x \in fanin(a)$ and $y = a$
(2) $x* \Rightarrow y*$, where $x = a$ and $y \in fanin(a)$
(3) $x* \Rightarrow y*$, where $x, y \in fanin(a)$ and $x = y$
(4) $x* \Rightarrow y*$, where $x, y \in fanin(a)$ and $x \neq y$

The arcs (1) - (3) are irrelevant to the isochronic fork relaxation. These orders are always guaranteed in the circuit. Therefore, the critical one is order (4) which assumes that the transition $x*$ propagates to gate a before it propagates along a path that triggers $y*$ and $y*$ reaches gate a. All such orders, if reversed by wire delays, are adversary paths in [3]. Fulfilling all the relations in (4) could guarantee the correctness of the circuit (equivalent to the no adversaries requirement in [3]). However, as will be seen later, this requirement still could be relaxed.

In this section we try to relax the transition orderings in case (4) as much as possible if no hazards would appear in the resulting STG. Relaxation of an arc $x* \Rightarrow y*$ in STG is depicted in Fig. 2 performed by Alg. 1.

Algorithm 1 Relax (STG, $x* \Rightarrow y*$)

```
for all b_i* ∈< x* do
   insert arc b_i* ⇒ y* in STG
end for
for all d_i* ∈ y*> do
   insert arc x* ⇒ d_i* in STG
end for
delete arc (x* ⇒ y*)
if x* ⇒ y* has a token then
   add a token on each newly inserted arc b_i* ⇒ y* and x* ⇒ d_i*
else
   for all newly inserted arcs b_i* ⇒ y* do
      if the arc b_i* ⇒ x* has a token then
         add a token on arc b_i* ⇒ y*
      end if
   end for
   for all newly inserted arcs x* ⇒ d_i* do
      if the arc y* ⇒ d_i* has a token then
         add a token on arc x* ⇒ d_i*
      end if
   end for
end if
STG = eliminate_redundant_arc (STG)
return STG
```

Fig. 2. Relaxation of arc $x* \Rightarrow y*$

From the STG view, relaxation changes one order assumption into a set of *weaker* order assumptions; from the circuit view, it removes the limitation that the transition $x*$ must reach the gate a before transition $y*$. After relaxation, in the SG corresponding to the new STG, more states are reachable. The original STG will be updated into the relaxed one if no new states contain potential hazards; if glitches appear in the newly added states then a timing constraint $x*$ must reach gate a before $y*$, denoted by $x* \Rrightarrow y*$, is added in the timing constraint set Rt and the arc $x* \Rightarrow y*$ is marked 'guaranteed' in the STG.

Each newly generated STG after relaxation is simplified before next relaxation by eliminating multiple and redundant arcs.

3) Hazard criterion: Whether the relaxation of an arc is acceptable or not depends on whether the newly introduced states cause glitches. A glitches is a premature transition, in which the output of a gate is enabled when it is expected to remain stable.

We first define the prerequisite event set of an event, the prerequisite event set of the i-th occurrence of event $a*$, $E_{pre}(a*/i) = \{z* : z* \in ({}^{<}a*/i)\}$. The prerequisite event set for each transition on output signal a is calculated before relaxation and is used to check the correctness after this relaxation has carried out.

When an arc $x* \Rightarrow y*$ of type (4) in a STG is relaxed, in the SG of resulting STG one of the four cases will be satisfied:

relaxation case1: $\forall s \in QR(a+)$, $f_{a\downarrow}(s) = FALSE$ and $\forall s \in QR(a-)$, $f_{a\uparrow}(s) = FALSE$.

relaxation case2: $\exists s \in QR(a+)$ such that $f_{a\downarrow}(s) = TRUE$ and $\forall s \in QR_i(a+)$ such that $f_{a\downarrow}(s) = TRUE$, $s(z) = 1$ if $z+ \in E_{pre}(a-/j)$; $s(z) = 0$ if $z- \in E_{pre}(a-/j)$, where $QR_i(a+)$ is followed by $ER_j(a-)$. Or similarly for $QR(a-)$, $f_{a\uparrow}$, $QR_i(a-)$ and $ER_j(a+)$.

If for all states s satisfy all the conditions in case 2, complementing signal x in s turns at least one clause c in the *disjunctive normal form* of $f_{a\uparrow}$ from *FALSE* to *TRUE* if $s \in QR(a-)$, or turns at least one clause c in *disjunctive normal form* of $f_{a\downarrow}$ from *FALSE* to *TRUE* if $s \in QR(a+)$ Then it is **relaxation case2.2** otherwise it is **relaxation case2.1**.

relaxation case3: $\exists s \in QR_i(a+)$ such that $f_{a\downarrow}(s) = TRUE$ and either $\exists z+ \in E_{pre}(a-/j)$ and $s(z) = 0$ or $\exists z- \in E_{pre}(a-/j)$ and $s(z) = 1$, where $QR_i(a+)$ is followed by $ER_j(a-)$. And, $\forall s$ that fulfill all conditions above, s' is the state obtained by complementing the value of signal x in s, then $s' \in ER_j(a-)$. Or similarly for $QR(a-)$, $f_{a\uparrow}$, $QR_i(a-)$ and $ER_j(a+)$.

relaxation case4: $\exists s \in QR_i(a+)$, $f_{a\downarrow}(s) = TRUE$ and either $\exists z+ \in E_{pre}(a-/j)$ and $s(z) = 0$ or $\exists z- \in E_{pre}(a-/j)$ and $s(z) = 1$, s' is the state by complementing the value of signal x in s and $s' \notin ER_j(a-)$where $QR_i(a+)$ is followed by $ER_j(a-)$. Or similarly for $QR(a-)$, $f_{a\uparrow}$, $QR_i(a-)$ and $ER_j(a+)$.

Fig. 3. Examples to the four relaxation cases; the dashed line indicates the state that fulfills all the conditions in each case

Relaxation case 1 is the case where relaxation does not cause any glitches. Relaxation case 2 and 3 are the situations

Algorithm 2 Expand (STG$_a$, NM, $f_{a\uparrow}$, $f_{a\downarrow}$)

```
while NM != NULL do
  x* ⇒ y* = find_strongest_arc (NM)
  if Check (Write_sg (Relax (STG, x* ⇒ y*)), f_a↑, f_a↓) == relaxation case 1 then
    STG = Relax (STG, x* ⇒ y*)
  else if Check (Write_sg (Relax (STG, x* ⇒ y*)), f_a↑, f_a↓) == relaxation case 2.1
  then
    if Check (Write_sg (STG' in relaxation 2.1), f_a↑, f_a↓) == relaxation case 1
    then
      STG = Relax (Relax (STG, x* ⇒ y*), x* ⇒ a*)
    else
      Rt = Rt ∪ {x* ⇒ y*}
      NM = NM - {x* ⇒ y*}
    end if
  else if Check (Write_sg (Relax (STG, x* ⇒ y*)), f_a↑, f_a↓) == relaxation case 2.2
  then
    if Check (Write_sg (STG" in relaxation 2.2), f_a↑, f_a↓) == relaxation case 1
    then
      Expand (STG', NM', f_a↑, f_a↓)
      Expand (STG", NM", f_a↑, f_a↓)
    else
      Rt = Rt ∪ {x* ⇒ y*}
      NM = NM - {x* ⇒ y*}
    end if
  else if Check (Write_sg (Relax (STG, x* ⇒ y*)), f_a↑, f_a↓) == relaxation case 3
  then
    if Check (Write_sg (STG" in relaxation 3), f_a↑, f_a↓) == relaxation case 1 then
      Expand (STG', NM', f_a↑, f_a↓)
      Expand (STG", NM", f_a↑, f_a↓)
    end if
  else
    Rt = Rt ∪ {x* ⇒ y*}
    NM = NM - {x* ⇒ y*}
  end if
end while
```

Fig. 4. Relaxation results in case2.1, case2.2 and case 3

that the gate is enable in some states in QR($a*$) but may not imply a glitch. Relaxation case 4 is the case where glitches will definitely appear. Case 2 occurs when one transition $x*$ which cannot be guaranteed to be acknowledged by the gate output transition $a*$ is relaxed to be one predecessor transition of $a*$. If $x*$ cannot cause $a*$ in any cases it is the case 2.1, otherwise it is the case 2.2. Case 3 occurs when transtion $x*$ triggers $a*$ but when the arc $x* \Rightarrow y*$ is relaxed, $y*$ could trigger $a*$ instead of $x*$. If case 2.1, 2.2 or 3 occurs then the STG will be modified as shown in Fig. 4 (if in case 2.2 or 3, the STG will be split into 2 sub-STGs, and each of them will be expanded recursively). All the new STG(s) after modification will be checked if they contain glitches. If

not the STG will be updated, otherwise, a timing constraint $x* \Rrightarrow y*$ will be added. Arcs marked with a # symbol in Fig. 4 are only used to indicate 'if in that case' (e.g. if in the case that $x*$ comes before $y*$) and will not be relaxed in the future steps (also not a timing constraint). The function that relaxes all possible arcs in a STG is listed in Alg. 2.

4) Optimal relaxation ordering: Different relative timing constraint sets might be derived if arcs are relaxed in different orders. This is shown in Fig. 5. Four different sets of timing constraints could guarantee the correctness of the circuit: $\{a+ \Rrightarrow b-,\ b+ \Rrightarrow c+\}$, $\{b+ \Rrightarrow a-,\ b+ \Rrightarrow c+\}$, $\{a+ \Rrightarrow c+,\ a+ \Rrightarrow b-\}$ and $\{a+ \Rrightarrow a+,\ b+ \Rrightarrow a-\}$. We prefer to generate the optimal one during the relaxation process rather than generate all of the cases and choose the best one. Exhaustion all relaxation orders implies a time complexity of $O(n!)$ w.r.t. the number of arcs to be relaxed and most of these relaxations lead to the same results.

Here, we consider the criterion for the weakest timing constraint set as the one where the tightest constraint in it is the loosest among all sets. The weakest constraint set could be generated by relaxing the tightest arc at each step. This will relax tighter arcs as much as possible before they become the necessary timing requirement to avoid entering the hazardous state. The standard of the tightness could be automatically determined by the topology of the arc in the circuit. For example, the path that crosses the environment is usually considered looser and the path which contains more levels is looser. This information is derived from the STG_{imp} and the function $find\_tightest\_arc(NM)$ returns the tightest $input \Rightarrow input$ arc in NM, the arc set whose orderings have not been guaranteed yet.

The top level algorithm to derive the weakest timing constraints for a SI circuit to work under the wire delay model is depicted in Alg. 3.

Fig. 5. Different timing constraints due to different relaxation ordering (timing constraints are denoted by &)

Algorithm 3 Deriving_timing_constraints(STG_{spec}, C)

```
STG_imp = Generate_STG_imp(STG_spec, C)
Rt = NULL
NM = NULL
MG = Decompse_into_MarkedGraph(STG_imp)
for all MG_j ∈ MG do
  for all signal a ∈ I ∪ O do
    STG_a = Deriving_local_STG(MG_j, a)
    NM = all the arcs between different input signals in STG_a
    Expand (STG_a, NM, f_a↑, f_a↓)
  end for
end for
return Rt
```

B. Delay padding to fulfill the dangerous timing constraints

When the relaxation is done, all of the generated timing constraints are changed into the pairwise delay constraints between a wire and a path by tracking back to the implementation specification STG_{imp} and then looking up the Circuit C. As an example the STG_{imp} and the circuit of a FIFO is presented in Fig. 7. The generated delay constraints using the technique in last section are shown in Tab. 1. The constraint in each row means that the delay of the wire in first column must be smaller than the sum of the delays in the third column. The circuit is guaranteed to work correctly if all of these delay constraints are fulfilled. Most constraints in table 1 are quite loose, which are considered to be fulfilled automatically; while, when some constraints are considered dangerous, delay padding to guarantee these orders (or other technique that could fix the order of two events) has to be carried out.

Fig. 6 shows the possible padding position (position 1-5) to guarantee the delay constraint that a wire from gate *g_1* to *g_4* should be faster than another path between these two gates. Padding on position 1, 3 or 5 (padding on wire), only delays transitions on one branch of a gate; while padding on position 2 or 4 (padding on gate) will delay all branches of a gate, which is equivalent to the increase of the delay of a gate. Padding on a gate could always fulfill one delay constraint without worsening other delay constraints, but might unnecessarily delay other branches in a fork; while padding on a wire has less performance penalty but might worsen another delay constraint if the wire that the delay padded on should be faster than another adversary path. A greedy padding policy is used which tries to pad the delay on position 1 if the corresponding wire does not in another delay constraint (in column 1), if it does then, tries to pad on position 3. In the worst case all of the wires in the adversary path are in some other delay constraints then pad on the position 2 could break this cyclic demand.

Due to the padding rule described above it could be guaranteed that all of the delay constraints could be fulfilled (padding on the last gate could always fulfill this delay constraint without worsening another, like the technique used in synthesis in [9]) and the optimal padding method to get the minimum performance penalty is that try to pad delays on wire near the destination gate of an adversary path such that this wire is not in the *fast path* of another delay constraint by looking up the delay constraint table such that this wire does not appear in the first column.

Fig. 6. Padding delay to guarantee the timing constraint

The constraints in Tab. 1 could be fulfilled by just delay one direction transitions thus introduces less performance

penalties, this could be done using the current starved delays. The performance penalties using different delays in FIFO example are shown in the next section.

Fig. 7. The STG_{imp} and circuit of a FIFO

TABLE I
LIST OF TIMING CONSTRAINTS

wire	$\leq$	adversary path
w15+		w14+, gate_0+, w4+
w14+		w3+, ENV, w17+
w3-		w5-, gate_2+, w7+, gate_L-, w14-
w6-		w9-, gate_3+, w10+, gate_4-, w12-,gate_L+, w17+,ENV, w5+
w5-		w1+, gate_Ro+, w2+, ENV, w6+
w8+		w1+, gate_Ro+, w2+, w6+, gate_2-, w7-, gate_L+, w8+, gate_3-, w10-, gate_4+, w12+, gate_L-, w14-, gate_0-, w4-, gate_Ai+, ENV, w16+, w13-, gate_L+, w17+, w1-, gate_Ro-, w2-, w9-
w9+		w6+, gate_2-, w7-, gate_L+, w17+, ENV, w8-
w13+		w11+, gate_4-, w12-, gate_L+, w17+, ENV, w1+, Ro+, w2+, ENV, w6+, gate_2-, w7-
w11-		w13-, gate_L+, w17+, ENV, w1-, Ro-, w2-, w9-, gate_3+, w10+

IV. EXPERIMENTS AND RESULTS

A. Comparison of timing constraints

The generated timing constraints are significantly weaker compared with the constraints implied by timing assumption proposed in [3], which is currently the weakest formally proved conditions. To the FIFO example, there are three $\leq$3-level (two wires and one gate in the adversary path) and four $\leq$5-level (three wires and two gate) timing constraints in timing assumption proposed in [3], which do not cross the environment. But only one and two respectively, generated by our method.

B. Delay penalty introduced by delay padding

The FIFO circuit is simulated under SPICE simulation using ASU PTM bulk CMOS model library (90nm to 32nm) [5] for testing the delay penalty due to the padding. Fig. 8 shows the delay penalty to eliminate all the glitches in one million gates scale (scale decides the maximum wire length) using different padding methods (buffer and one-direction current-starved delay). The delays are inserted to *just* counter the maximum wire length delay, the environment is assumed to be *zero* delay and the delay penalty is calculated as the maximun latency increase in the slowest STG cycle.

Fig. 8. Performance penalty under different delay padding methods

V. CONCLUSIONS

Are the timing constraints produced by our method necessary? The answer is it is not guaranteed. One example is the circuit in Fig. 5. There are four sets of timing constraints that could guarantee the correctness of the circuit thus *none of them* is necessary. Our method just generates the weakest one among all solutions w.r.t. a certain criterion.

Speed independent design suggests a good paradigm to tolerate process variations but the more and more sever wire delays and process variations also challenge the fundamental assumptions of the speed independent design. This paper proposes a method that fixes only few potential hazardous places in the speed independent circuits to make sure that speed independent circuits could work correctly under the wire delay dominance and in the variation age.

REFERENCES

[1] A.J. Martin, The Limitations to Delay-Insensitivity in Asynchronous Circuits *Proc. Sixth MIT Conf. Advanced Research in VLSI*, pp. 263-278, Cambridge, Mass.: MIT Press, 1990.

[2] N. Sretasereekul, and T. Nanya, Eliminating isochronic-fork constraints in quasi-delay-insensitive circuits. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*. ASP-DAC '01. 437-442.

[3] S. Keller, M. Katelman, A. J. Martin, A Necessary and Sufficient Timing Assumption for Speed-Independent Circuits, *Asynchronous Circuits and Systems, International Symposium on*, 2009.

[4] S. H. Unger, Hazards, Critical Races, and Metastability. *IEEE Transactions on Computers*, vol. 44, no. 6, pp. 754-768, June 1995.

[5] http://ptm.asu.edu/

[6] O. Roig, J. Cortadella, E. Pastor: Hierarchical gate-level verification of speed-independent circuits. *ASYNC*, 1995.

[7] J. Cortadella, M. Kishinevsky, L. Lavagno and A. Yakovlev, Deriving Petri Nets from Finite Transition Systems *IEEE Transactions on Computers*, vol. 47, pp. 859-882, 1998.

[8] T.-A. Chu, Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications. PhD thesis, MIT, June 1987.

[9] L. Lavagno, K. Keutzer and A. Sangiovanni-Vincentelli, Algorithms for synthesis of hazard-free asynchronous circuits, *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 1991.

[10] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002.