A Necessary and Sufficient Timing Assumption for Speed-Independent Circuits

Sean Keller*, Michael Katelman†, Alain J. Martin*
*Department of Computer Science
California Institute of Technology
Pasadena, CA 91125, USA
{sean,alain}@async.caltech.edu
†Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
katelman@uiuc.edu

Abstract

This paper presents a proof that the adversary path timing assumption is both necessary and sufficient for correct SI circuit operation. This assumption requires that the delay of a wire on one branch of a fork be less than the delay through a gate sequence beginning at another branch in the same fork. Both the definition of the timing assumption and the proof build on a general, formal notion of computation given with respect to production rule sets. This underlying framework can be used for a variety of proof efforts or as a basis for defining other useful notions involving asynchronous computation.

1. Introduction

Asynchronous logic can be effectively engineered within a variety of different frameworks, *e.g.* via quasi-delay insensitive (QDI) [1], [2], [3] or speed-independent (SI) circuits [4], [5]. Across the various frameworks there are clearly many important differences, but these frameworks also share certain issues that seem to be inherent to asynchronous circuit design; in particular, the notion of *forks*. For example, the class of SI circuits is characterized as the set of circuits that are functionally correct regardless of gate and wire delays, *except* at forks; similarly, forks play a crucial role in the context of QDI circuits. In fact, if no delay assumptions are made about forks, then the resulting delay-insensitive (DI) circuits are extremely limited in functionality [6]. Moreover, for many asynchronous logic frameworks, the relative delays through fork branches form the basis of all timing assumptions and the corresponding timing closure. This suggests that in order to gain insight into the exact similarities and differences between these frameworks, it may be fruitful to compare their timing assumptions.

Making such comparisons can be difficult because different frameworks use different terms and mathematical constructions; a mathematical setting in which common terms are used to describe all of the timing assumptions is required. Therefore, towards clarifying the nature of forks across multiple asynchronous circuit frameworks, this paper first formally defines a notion of asynchronous computation and then upon that defines a set of well-known fork-related timing assumptions. Using this foundation, the paper then proves that one such assumption, the *adversary path timing assumption* [3], is necessary and sufficient for proper SI circuit operation. Moreover, this proof suggests that the adversary path is the longest such path and hence leads to the *weakest* SI timing constraint.

The foundation starts with the structure given by *production rule sets* (PRS). This is not crucial: *any number of systems can be used instead*. However, PRS are structured in such a way that clearly exposes how forks and hazards [5], [7] interact. Moreover, PRS can be used to model arbitrary switching networks, and therefore can be used to examine important properties of circuits generated within a wide range of asynchronous design methodologies. In addition, the proof is just one example of how the formalization can be used. It can also serve as a foundation for other proofs or definitions, and it can even be used as a basis for computerized proofs.

Each of the following technical sections contains both an informal overview of main concepts, with examples derived from the circuit depicted in Figure 1 (a closed variant of a circuit used in [6]), as well as thoroughly developed mathematical details. The details add rigor and some subtle insights, but the main ideas and notation are presented at a higher level. The organization of the paper is as follows. Section 2 reviews production rule sets and defines a set of structural constraints that are assumed throughout. Section 3 formally defines a notion of computation with respect to PRS. Then, upon this notion of computation, Section 4 defines the relevant timing assumptions for DI, QDI, and SI circuits. The proof and a discussion of its implications are given in Section 5. Section 6 reviews related work, and Section 7 concludes with a summary and a discussion of several assumptions and limitations of this work.

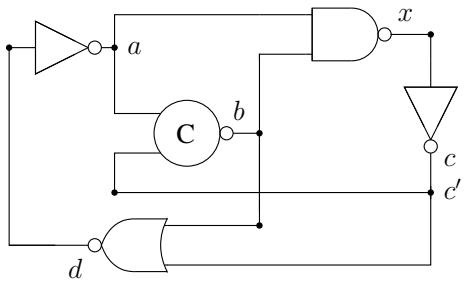

Figure 1. **Closed simple buffer.**

2. PRS Structural Constraints

Section 2.1 begins by reviewing the traditional definition of PRS [1]. Section 2.2 adds a set of structural constraints that facilitate the definition of a formal notion of computation and timing assumptions in Sections 3 and 4, respectively. These structural constraints ultimately result in a set of "legal" production rule sets, which are called *proper*.

2.1. PRS

This section reviews a few of the basic terms from [1]. The mapping from PRS to CMOS transistor networks, used in a number of figures throughout this paper, also comes from [1]. Additionally, in order to simplify the exposition, this paper assumes a fixed set V of variables from which the PRS draw node names; $T_{\mathbb{B}}(V)$ denotes the set of Boolean expressions over variables V.

Definition 2.1. A *production rule* is any triple

$$(g, x, d) \in T_{\mathbb{B}}(V) \times V \times \{\uparrow, \downarrow\}$$

and is typically denoted $g \mapsto xd$.

Definition 2.2. A *production rule set* (PRS) is any finite set of production rules.

The basic intuition for a production rule $g \mapsto x\uparrow$ is that g is a *sufficient* condition to enable the pull-up network in the gate associated with x. This means that the entire condition for the pull-up network can be spread out across multiple production rules $g_i \mapsto x\uparrow$. For example, one of the structural constraints that this paper enforces, without loss of generality, is that each variable x is defined by *exactly two production rules*, $g^+ \mapsto x\uparrow$ and $g^- \mapsto x\downarrow$.

2.2. "Proper" PRS

The definitions in this section serve to impose extra structure on PRS. This added structure facilitates a straightforward definition of computation in Section 3 and the mapping of computation to physical circuits; furthermore these constraints simplify the definition of timing assumptions in Section 4 and the proof in Section 5.

Definition 2.3. Let $\mathcal{P}$ be a PRS and $x \in V$, the x *operator on* $\mathcal{P}$, denoted $\mathcal{O}_x$, is defined such that for all $g \mapsto x'd \in \mathcal{P}$

$$g \mapsto x'd \in \mathcal{O}_x \Leftrightarrow x' = x$$

Definition 2.4. Let $\mathcal{P}$ be a PRS. $\mathcal{P}$ *has simple operators* if and only if all $\mathcal{O}_x$ are such that

$$\mathcal{O}_x = \{g^+ \mapsto x\uparrow, g^- \mapsto x\downarrow\}$$

and g^+ and g^- are in *disjunctive normal form*.

Definition 2.5. Let $\mathcal{P}$ be a PRS with simple operators. $\mathcal{O}_y$ is called a *wire* if and only if

$$\mathcal{O}_y = \{x \mapsto y\uparrow, \neg x \mapsto y\downarrow\}$$

for some $y \in V$. An operator $\mathcal{O}_x$ is called a *gate* if it is not a wire.

The majority of structural constraints for a proper PRS simply enforce a regular forking structure. These requirements essentially guarantee a one-to-one correspondence between the branches of a fork and *wire operators*. In order to define and force these and other structural requirements, it is necessary to have a clean way of expressing variable sharing within and between the operators, as such sharing can imply forking. This is formalized beginning with the function π, which counts the number of occurrences of a specified variable in the guard of a production rule.

Definition 2.6. $\pi : V \times T_{\mathbb{B}}(V) \longrightarrow \mathbb{N}$:

$$\begin{aligned}
\pi(x, x) &= 1 \\
\pi(x, x') &= 0 \qquad \textit{if } x' \neq x \\
\pi(x, g_1 \wedge g_2) &= \pi(x, g_1) + \pi(x, g_2) \\
\pi(x, g_1 \vee g_2) &= \pi(x, g_1) + \pi(x, g_2) \\
\pi(x, \neg g_1) &= \pi(x, g_1)
\end{aligned}$$

Using Definition 2.6, PRS variables are related to each other by constructing a directed multi-graph with a node for every variable and a directed weighted-edge between pairs of variables.

Definition 2.7. Let $\mathcal{P}$ be a PRS. Associate to $\mathcal{P}$ a directed multi-graph $(V, E : V \times V \longrightarrow \mathbb{N})$ where

$$E(x, x') = \sum_{g \mapsto x'd \in \mathcal{O}_{x'}} \pi(x, g)$$

The most important information encoded by this graph is a matching of input variables to the output variable of each gate. This is expressed via the following $\longrightarrow$ relation.

Definition 2.8. Let $\mathcal{P}$ be a PRS. With respect to $\mathcal{P}$, $\longrightarrow \subseteq V \times V$ is a binary relation defined such that for all x, x'

$$x \longrightarrow x' \Leftrightarrow E(x, x') > 0$$

Figure 2 expands the NAND gate from Figure 1 and adds two wires, $\mathcal{O}_{a'}$ and $\mathcal{O}_{a''}$. This expansion illustrates several definitions, *e.g.* $E(a'', x) = E(a, x) = 1$, $E(b, x) = 2$, $a \longrightarrow a'$, and $a' \longrightarrow a''$. The $\longrightarrow$ relation is employed extensively, and in many cases the following notational conventions are used: $\cdot \longrightarrow x'$, which means the *set* $\{x \mid x \longrightarrow x'\}$, and $x' \longrightarrow \cdot$, which means the *set* $\{x \mid x' \longrightarrow x\}$. With respect to Figure 2, $\cdot \longrightarrow x = \{a, a'', b\}$, and $b \longrightarrow \cdot = \{x\}$. This notion usefully extends to multiple arrows and multiple dots; *e.g.* $x \longrightarrow \cdot \longrightarrow x'$ or $\cdot \longrightarrow \cdot \equiv\!\!\longrightarrow$.

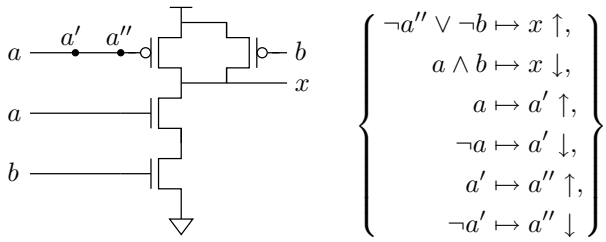

Figure 2. **CMOS NAND gate and wires.**

The existence of composed wires is equivalent to the statement that there exists wires $\mathcal{O}_y$ and $\mathcal{O}_{y'}$, $y \neq y'$, such that $y \longrightarrow y'$, *e.g.* in Figure 2, $a' \longrightarrow a''$. This sort of composition is not allowed in a proper PRS; and, similarly, gate-to-gate connections are also disallowed, *e.g.* in Figure 1, the variable a acts as both the output of the inverter and an input of the NAND gate. Therefore, between gates, a signal must go through *exactly one* wire.

Definition 2.9. Let $\mathcal{P}$ be a PRS. $\mathcal{P}$ has *no wire-to-wire connections* if and only if for all pairs of wires $\mathcal{O}_y, \mathcal{O}_{y'}$, $(y, y') \notin\!\!\longrightarrow$.

Definition 2.10. Let $\mathcal{P}$ be a PRS. $\mathcal{P}$ has *no gate-to-gate connections* if and only if for all pairs of gates $\mathcal{O}_x, \mathcal{O}_{x'}$, $(x, x') \notin\!\!\longrightarrow$.

This leaves the possibility of implicit forking through sharing of wire variables across different gates; which can be removed by enforcing that $\mathcal{P}$ has *explicit inter-operator forks*.

Definition 2.11. Let $\mathcal{P}$ be a PRS satisfying the conditions of Definitions 2.9 – 2.10. $\mathcal{P}$ has *explicit inter-operator forks* if and only if for all wires $\mathcal{O}_y$, $|y \longrightarrow \cdot| = 1$.

Figure 3 transforms an implicit inter-operator fork from Figure 1 into an explicit inter-operator fork by connecting inverter $\mathcal{O}_a$ to two new wires $\mathcal{O}_{a_1}$ and $\mathcal{O}_{a_2}$. Nevertheless, variable sharing can occur within a gate, *e.g.* this happens in Figure 2, where $E(b, x) = 2$. This leads to the final structural constraint on forks.

Definition 2.12. Let $\mathcal{P}$ be a PRS satisfying the conditions of Definitions 2.9 – 2.10. $\mathcal{P}$ has *explicit intra-operator forks*

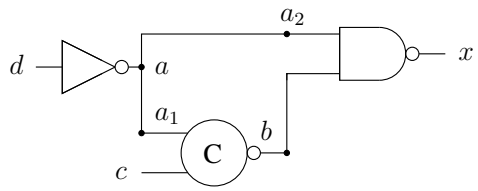

Figure 3. **Explicit inter-operator fork.**

if and only if for all wires $\mathcal{O}_y$, if $y \longrightarrow x$ then $E(y, x) = 1$.

Figure 4 further expands Figure 3 by making explicit the NAND gate intra-operator forks. This necessitates the addition of another new wire, $\mathcal{O}_{a_3}$, to inverter $\mathcal{O}_a$ and two new wires $\mathcal{O}_{b_1}$ and $\mathcal{O}_{b_2}$ to C-element $\mathcal{O}_b$. Definitions 2.9 – 2.12 ensure that at the switch level, there is a one-to-one correspondence between gate interconnections and wires.

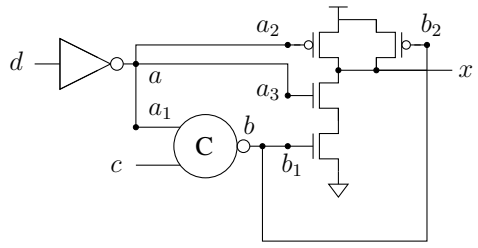

Figure 4. **Explicit intra-operator fork.**

Making *inter-operator* forks explicit is commonplace and essential for a discussion of asynchronous circuits. Making *intra-operator* forks explicit is less typical but not unprecedented (see [8]); they are exposed for completeness in Section 4 on timing assumptions. In what follows, a properly structured PRS is closed and is considered to have all of the above properties.

Definition 2.13. Let $\mathcal{P}$ be a PRS. $\mathcal{P}$ *is closed* if and only if for all $x \in V$; $\cdot \longrightarrow x \neq \emptyset$ and $x \longrightarrow \cdot \neq \emptyset$.

Definition 2.14. Let $\mathcal{P}$ be a PRS. $\mathcal{P}$ *proper* if and only if $\mathcal{P}$ satisfies the conditions of Definition 2.4 and Definitions 2.9–2.13.

3. PRS Semantics

This section defines a mapping from PRS to legal *computations*, where *computations* are legal if they fall within the set of dynamic behaviors defined by a circuit. The formalization treats PRS as a set of concurrent processes with each gate and wire acting individually. The main ideas as well as several examples are presented in Section 3.1, followed by further details in Section 3.2.

3.1. Overview

Conceptually, the definition of computation given in this section treats gates and wires as *independent processes*. These processes are continuously sensitive to the current state of all nodes named in the guards of the corresponding pair of production rules. For example, the expanded NAND gate shown in Figure 4 is treated as a process that is sensitive to the state of four nodes: a_2, a_3, b_1, b_2. At any given "step" in the computation, each process can either (a) act on the current state of its inputs by transitioning its target node appropriately, or (b) delay a pending transition to a future step. There is also a third possibility, (c): a gate can express a pending *hazard*.

Ignoring hazards for the moment, the *state* of the circuit nodes is encoded as a function, $\chi : V \longrightarrow \{\mathtt{F}, \mathtt{T}\}$, which maps nodes to logical values. For example, $\chi(a_2) = \mathtt{T}$ means that the current state of node a_2 acts as logical true. Now, consider again the NAND gate and a state

$$\chi(a_2) = \mathtt{T}, \chi(a_3) = \mathtt{T}, \chi(b_1) = \mathtt{T}, \chi(b_2) = \mathtt{T}, \chi(x) = \mathtt{T}.$$

A computation "step" takes the *current state*, χ, to a *new state*, χ'. Corresponding to cases (a) and (b) above, there are two alternatives for x in χ'. Either (a) the pending transition gets expressed and $\chi'(x) = \mathtt{F}$, or (b) the transition is delayed and $\chi'(x) = \chi(x) = \mathtt{T}$.

For every gate, there is also a specific set of undesirable states that expose its non-digital and non-atomic nature. These hazardous states generate uncertainty in the logic value of the gate output. As such, subsequent gates may individually interpret the value as either T, F, or undefined. These possibilities necessitate further enrichment of state beyond $\chi : V \longrightarrow \{\mathtt{F}, \mathtt{T}\}$. First, for hazards to be explicitly manifested, the co-domain of χ is expanded to the set $\{\mathtt{F}, \mathtt{X}, \mathtt{T}\}$, so that χ becomes a function $\chi : X \longrightarrow \{\mathtt{F}, \mathtt{X}, \mathtt{T}\}$. Second, the state is enriched so that it becomes a pair (χ, I) with χ as above and $I \subseteq V$, where I is a set containing all nodes with pending hazards. That is, $x \in I$ implies that case (c) is a valid option, so $\chi'(x) = \mathtt{X}$ is possible in some future computation step.

3.2. Formalization

The definition of computation is given as a binary relation on *execution states*. From this point onward, denote by $\mathbb{B}$ the structure with elements $\{\mathtt{F}, \mathtt{X}, \mathtt{T}\}$ and functions $\neg, \wedge, \vee$.

Definition 3.1. An *execution state* is any pair

$$(\chi : V \longrightarrow \mathbb{B}, I \subseteq V).$$

In order to define inference rules for generating the next system state χ' from the current state χ, it is useful to have formal notions describing the current "state" of a gate. Intuitively, if χ is such that a gate is being *pulled up*, then the gate is allowed to transition to T. Similarly, if the gate is being *pulled down*, then it is allowed to transition to F and if there is a pending hazard, it is allowed to transition to X. The following definitions formalize the various sensitivities of a gate.

Definition 3.2. Let $\chi : V \longrightarrow \mathbb{B}$ and $g \in T_{\mathbb{B}}(V)$. $\chi(g)$ denotes the extension of χ to Boolean expressions.

Definition 3.3. Let $\chi : V \longrightarrow \mathbb{B}$ and let $\mathcal{O}_x$ be a gate defined such that

$$\mathcal{O}_x = \{g^+ \mapsto x\uparrow, g^- \mapsto x\downarrow\}.$$

- $A^{\uparrow}_{\chi}$ is a predicate on gates denoting that $\mathcal{O}_x$ is currently being *pulled up* with respect to χ, *i.e.*

$$A^{\uparrow}_{\chi}(\mathcal{O}_x) \Leftrightarrow \chi(g^+) = \mathtt{T} \text{ and } \chi(g^-) = \mathtt{F}.$$

- $A^{\downarrow}_{\chi}$ is a predicate on gates denoting that $\mathcal{O}_x$ is currently being *pulled down* with respect to χ, *i.e.*

$$A^{\downarrow}_{\chi}(\mathcal{O}_x) \Leftrightarrow \chi(g^+) = \mathtt{F} \text{ and } \chi(g^-) = \mathtt{T}.$$

- $A^{\updownarrow}_{\chi}(\mathcal{O}_x)$ is a predicate on gates denoting that $\mathcal{O}_x$ is *interfering*, or *shorted*, with respect to χ, *i.e.*

$$A^{\updownarrow}_{\chi}(\mathcal{O}_x) \Leftrightarrow \chi(g^+) = \mathtt{T} \text{ and } \chi(g^-) = \mathtt{T}.$$

- $A^{\bullet}(\mathcal{O}_x)$ is a predicate on gates, denoting that $\mathcal{O}_x$ is being *invalidated* with respect to χ, *i.e.*

$$A^{\bullet}_{\chi}(\mathcal{O}_x) \Leftrightarrow \chi(g^+) = \mathtt{X} \text{ or } \chi(g^-) = \mathtt{X}.$$

For the moment let us ignore how the I set is computed and just assume that any pending hazard is contained in I. The semantics allows for the state of any operator output to either change or to stay the same. A state change from χ to a state χ' must satisfy the following property: *for all* $x \in V$ *such that* $\chi(x) \neq \chi'(x)$:

$$\chi'(x) = \mathtt{T} \Rightarrow A^{\uparrow}_{\chi}(\mathcal{O}_x)$$
$$\chi'(x) = \mathtt{F} \Rightarrow A^{\downarrow}_{\chi}(\mathcal{O}_x)$$
$$\chi'(x) = \mathtt{X} \Rightarrow x \in I.$$

As there are *many such* χ' in general, there are many possible *next states*; this is a reflection of the natural, per-gate concurrency that is expressed in the above constraints whenever $\chi'(x) \neq \chi(x)$.

There are two varieties of hazards that can occur in asynchronous circuits: interferences and instabilities. The first type of hazard, *interference*, occurs when a gate is being shorted; *e.g.* the NAND gate of Figure 4, defined by production rules

$$\{\neg a_2 \vee \neg b_2 \mapsto x\uparrow, a_3 \wedge b_1 \mapsto x\downarrow\},$$

exhibits an interference when both guards evaluate to true, such as in a state where

$$\chi(a_2) = \mathtt{F}, \chi(a_3) = \mathtt{T}, \chi(b_1) = \mathtt{T}, \chi(b_2) = \mathtt{F}.$$

Definition 3.4. Let $\chi : V \longrightarrow \mathbb{B}$ and $\mathcal{O}_x$ a gate. $\mathcal{O}_x$ is *interfering* w.r.t. χ if and only if $A^{\updownarrow}_{\chi}(\mathcal{O}_x)$.

The second type of hazard is *unstable* behavior. This occurs when, at some state (χ, I), a gate $\mathcal{O}_x$ is enabled to transition (*i.e.* there exists a legal execution step to a state (χ', I') where $\chi(x) \neq \chi'(x)$) but does not transition in the *actual* step to (χ', I') (*i.e.* $\chi(x) = \chi'(x)$), *and* the inputs to $\mathcal{O}_x$ change when going from (χ, I) to (χ', I') in such a way that $\mathcal{O}_x$ is disabled from transitioning in the following step. This unstable behavior captures some of the non-atomic properties of gates in real circuits. If a gate begins to transition towards one rail, but is cut-off before completing this transition, the output of the gate may be interpreted individually by subsequent transistors as either T, F, or as a non-Boolean value.

Taking the NAND again as an example, it is *enabled* in the state $(\chi, \emptyset)$ with

$$\chi(a_2) = \mathtt{T}, \chi(a_3) = \mathtt{T}, \chi(b_1) = \mathtt{T}, \chi(b_2) = \mathtt{T}, \chi(x) = \mathtt{T}$$

in that there exists a legal step $\chi'(x) \neq \chi(x)$, $\chi'(x) = \mathtt{F}$. However, suppose that instead of this transition happening, only the gate's *inputs* change, so that χ' is given by

$$\chi'(a_2) = \mathtt{F}, \chi'(a_3) = \mathtt{F}, \chi'(b_1) = \mathtt{T}, \chi'(b_2) = \mathtt{T}, \chi'(x) = \mathtt{T}.$$

This gate is no longer enabled in the sense that during the next step, say to (χ'', I''), x cannot transition to the other stable value, *i.e.* $\chi''(x) = \mathtt{F}$ is impossible.

Definition 3.5. Let $\chi, \chi' : V \longrightarrow \mathbb{B}$ and $\mathcal{O}_x$ a gate. $\mathcal{O}_x$ is *unstable* w.r.t. χ, χ' if and only if

$$A^{\uparrow}_{\chi}(\mathcal{O}_x),\ \chi'(x) \neq \mathtt{T}, \text{ and } \neg A^{\uparrow}_{\chi'}(\mathcal{O}_x); \text{ or}$$
$$A^{\downarrow}_{\chi}(\mathcal{O}_x),\ \chi'(x) \neq \mathtt{F}, \text{ and } \neg A^{\downarrow}_{\chi'}(\mathcal{O}_x).$$

The I set tracks all pending hazards, so that in a "step" from (χ, I) to (χ', I'), it must be ensured that I' contains (a) all *interferences* with respect to χ', as well as (b) all *instabilities* with respect to χ, χ'. In addition to these two hazard *origination* events, X values must also be allowed to propagate. This is formalized by creating two auxiliary sets I^+ and I^-. The I^+ set simply accumulates all of the new interferences and instabilities generated in going from χ to χ', and the I^- set includes all variables that have transitioned. The set $I \setminus I^-$ is then used to allow unresolved hazards to persist from I to I'.

Definition 3.6. Let $\chi, \chi' : V \longrightarrow \mathbb{B}$; the set of *new potential hazards* with respect to χ, χ', denoted $I^+_{\chi,\chi'}$ is defined such that

$$u \in I^+_{\chi,\chi'} \Leftrightarrow \mathcal{O}_u \text{ is unstable w.r.t. } \chi, \chi',$$
$$\mathcal{O}_u \text{ is interfering w.r.t. } \chi', \text{ or } A^{\bullet}_{\chi'}(\mathcal{O}_u).$$

Similarly, the set of *non-persisting potential hazards*, denoted $I^-_{\chi,\chi'}$, is defined such that

$$u \in I^-_{\chi,\chi'} \Leftrightarrow \chi'(u) \neq \chi(u).$$

Definition 3.7. Let $\mathcal{P}$ be a proper PRS. The *computation step* relation, $\rightrightarrows$, is a binary relation on *execution states* defined such that $(\chi, I) \rightrightarrows (\chi', I')$ if and only if for all $x \in V$ with $\chi(x) \neq \chi'(x)$:

$$\chi'(x) = \mathtt{T} \Rightarrow A^{\uparrow}_{\chi}(\mathcal{O}_x)$$
$$\chi'(x) = \mathtt{F} \Rightarrow A^{\downarrow}_{\chi}(\mathcal{O}_x)$$
$$\chi'(x) = \mathtt{X} \Rightarrow x \in I,$$

and $I' = I^+_{\chi,\chi'} \cup I \setminus I^-_{\chi,\chi'}$.

Definition 3.8. Let $\mathcal{P}$ be a proper PRS and $\vec{\sigma} = \langle \sigma_1, \sigma_2, \dots \rangle$ be an infinite sequence of states. $\vec{\sigma}$ is a *legal execution sequence*, if and only if for all $i \geq 1$, $\sigma_i \ \rightrightarrows \ \sigma_{i+1}$.

In what follows, computations are restricted so as to satisfy a few important sensibility requirements. Such a computation assumes (a) that the reset state is free of interferences, instabilities, and X values, and (b) that the reset state initializes forks with the same value on every branch. The restriction on fork branches simplifies several timing assumptions given in Section 4.

Definition 3.9. Let $\mathcal{P}$ be a proper PRS and $\vec{\sigma} = \langle \sigma_1, \sigma_2, \dots \rangle$ an execution sequence. σ_1 is called the *reset state*.

Definition 3.10. Let $\mathcal{P}$ be a proper PRS and $\vec{\sigma}$ an execution sequence with reset state $\sigma_1 = (\chi_1, I_1)$. $\vec{\sigma}$ is *proper* if:

- for all x, $\chi_1(x) \neq \mathtt{X}$; and $I_1 = \emptyset$; and
- for all gates $\mathcal{O}_x$, for all $y, y' \in x \longrightarrow \cdot$, $\chi_1(y) = \chi_1(y')$.

Lastly, it is useful to extend the notions of stability and non-interference beyond a single sequence.

Definition 3.11. Let $\mathcal{P}$ be a proper PRS and σ_1 a reset state. $\mathcal{P}, \sigma_1$ is stable and non-interfering if and only if all proper executions σ with reset state σ_1 are stable and non-interfering.

4. Timing

Reaching a timing closure for an asynchronous system tends to be considerably easier to achieve than for a similar synchronous design. Even so, as CMOS evolves and becomes ever more varied, and as entirely new paradigms are targeted, certain assumptions about timing become harder to satisfy [3]. This section gives formal meaning to the terms used to discuss common timing assumptions made for asynchronous circuits and then uses these terms to provide concrete definitions for DI, QDI, and SI systems.

4.1. Transition Causality

An important concept used to reason about the sequencing of transitions is the notion of *acknowledgment*. Acknowledgment embodies the causal relationship between the current inputs of an operator, say $\mathcal{O}_x$, and a *transition* in the state of x, *e.g.* from $\chi(x) = \mathtt{T}$ to $\chi'(x) = \mathtt{F}$. This paper leverages the fact that all guard expressions of a proper PRS are in disjunctive normal form in order to say that each guard variable in a true-valued conjunctive clause is acknowledged when the target variable transitions.

Definition 4.1. Let $\vec{\sigma}$ be a proper execution sequence. Associate to $\vec{\sigma}$ an *acknowledgment relation*,

$$\leftrightarrows \subseteq V \times \mathbb{N} \times V,$$

and write $x \leftrightarrows_i x'$ when $(x, i, x') \in \leftrightarrows$. The relation is defined inductively such that

(a) $x \leftrightarrows_i x'$ if, letting

$$\mathcal{O}_{x'} = \{c_1 \vee \cdots \vee c_m \mapsto x' \uparrow, d_1 \vee \cdots \vee d_n \mapsto x' \downarrow\}$$

either

- $\chi_i(x') \neq \mathtt{T}$, $\chi_{i+1}(x') = \mathtt{T}$, and $\pi(x, c_j) > 0$ for some c_j such that $\chi_i(c_j) = \mathtt{T}$; or
- $\chi_i(x') \neq \mathtt{F}$, $\chi_{i+1}(x') = \mathtt{F}$, and $\pi(x, d_j) > 0$ for some d_j such that $\chi_i(d_j) = \mathtt{T}$;

(b) $x \leftrightarrows_i x'$ if $\mathcal{O}_{x'}$ is a wire with $x \longrightarrow x'$, and for some $x'' \in x \longrightarrow \cdot$, $x \leftrightarrows_i x''$; $\chi_{i+1}(x') = \chi_i(x)$; and letting j be the largest index less than i such that $y \leftrightarrows_j x$ for some y, $x \not\leftrightarrows_k x'$ for all $j < k < i$.

Condition (a) formalizes the well-known definition of acknowledgment as a causal relationship between transitions [6], and it extends the definition by allowing wires to acknowledge gates and gates to acknowledge wires. Condition (b) further extends acknowledgment to handle *inconsistencies* that can occur at certain forks. As an example, consider Figure 5. This figure completely exposes all forks from a segment of the circuit from Figure 1. Notice that gates and wires inherently "hold" state, so b is automatically *staticized*. Now, consider a proper execution sequence $\vec{\sigma}$, where σ_i is specified by Figure 5. In this state, the inverter $\mathcal{O}_a$ is enabled to transition, as are the wires $\mathcal{O}_{a_2}$, $\mathcal{O}_{a_3}$, $\mathcal{O}_{b_1}$, and $\mathcal{O}_{b_2}$. If the inverter output transitions but the wires do not, then $\chi_{i+1} = \chi_i[a \mapsto \mathtt{T}]$, and by condition (a) of acknowledgment, $d_1 \leftrightarrows_i a$. Continuing with this example, suppose that the $\mathcal{O}_{a_1}$ wire transitions next, yielding $\chi_{i+2} = \chi_{i+1}[a_1 \mapsto \mathtt{T}]$. By condition (a) of acknowledgment, $a \leftrightarrows_{i+1} a_1$, and by condition (b) $a \leftrightarrows_{i+1} a_2$ and $a \leftrightarrows_{i+1} a_3$. In some sense O_{a_2} and O_{a_3} skipped a transition (legally), and condition (b) maintains a consistent notion of acknowledgment. Furthermore, acknowledgment "chains" give rise to a transitive version of acknowledgment.

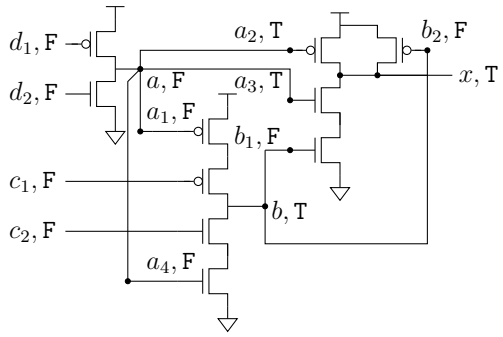

Figure 5. **Simple buffer segment; at state** $\sigma_i = (\chi_i, \emptyset)$.

Definition 4.2. Let $\vec{\sigma}$ be a proper execution sequence. Associate to $\vec{\sigma}$ a relation

$$\leftrightarrows^+ \subseteq V \times \mathbb{N} \times \mathbb{N} \times V,$$

and write $x \leftrightarrows^+_{[m,n]} x'$ when $(x, m, n, x') \in \leftrightarrows^+$. The relation is defined inductively such that

- if $x \leftrightarrows_i x'$ then $x \leftrightarrows^+_{[i,i+1]} x'$;
- if $x \leftrightarrows^+_{[m,n]} x'$ and $\chi_{n+1}(x') = \chi_n(x')$, then $x \leftrightarrows^+_{[m,n+1]} x'$;
- if $x \leftrightarrows^+_{[m,n]} x'$ and $x' \leftrightarrows_n x''$, then $x \leftrightarrows^+_{[m,n+1]} x''$.

4.2. Timing Assumptions

All of the timing assumptions presented in this paper involve forks. Furthermore, these assumptions are defined and applied in terms of general n-way forks, as opposed to simply binary forks. The first such timing assumption is frequently overlooked because it places restrictions on forks internal to gates. These intra-operator forks are usually concealed by sharing variables across distinct conjunctive clauses within operator guard expressions, but they are made explicit by disallowing shared variables in every proper PRS. These forks are intentionally exposed, because they exist in real circuits, and they accurately account for a number of analog circuit constraints [9], [8].

The strong intra-operator fork assumption states that if any branch of a fork emanating from gate $\mathcal{O}_x$ has been acknowledged by a wire leading to another gate, say $\mathcal{O}_{x'}$, then all branches of the fork leading to $\mathcal{O}_{x'}$ have been acknowledged. This assumption is part of the standard gate-based digital circuit abstraction; *e.g.* in CMOS circuits, it abstracts away details such as switching slew rates and relative transistor strengths. Additionally, it greatly simplifies the execution model, as hazard-free execution sequences are entirely within the *digital* realm; *i.e.* at every step each variable can be interpreted as either T or F.

Definition 4.3. Let $\vec{\sigma}$ be a proper execution sequence. $\vec{\sigma}$ satisfies the *strong intra-operator fork timing assumption* if

and only if for all pairs of gates $\mathcal{O}_x, \mathcal{O}_{x'}$ and every index i;

if $x \leftrightarrows_i y$ for some $y \in x \longrightarrow \cdot \longrightarrow x'$, then
$x \leftrightarrows_i y'$ for *all* $y' \in x \longrightarrow \cdot \longrightarrow x'$.

Consider a proper execution sequence $\vec{\sigma}$, where σ_i is specified by Figure 5, and the execution step where $\chi_{i+1} = \chi_i[a_2 \mapsto \mathtt{F}]$. This execution step does not satisfy the strong intra-operator fork timing assumption as $a \leftrightarrows_i a_2$ but not $a \leftrightarrows_i a_3$.

The next assumption is nearly identical but constrains forks branching out to distinct operators.

Definition 4.4. Let $\vec{\sigma}$ be a proper execution sequence. $\vec{\sigma}$ satisfies the *strong inter-operator fork timing assumption* if and only if for every gate $\mathcal{O}_x$ and index i

if $x \leftrightarrows_i y$, then for all x' such that $x \longrightarrow \cdot \longrightarrow x' \neq \emptyset$
$x \leftrightarrows_i y'$ for *some* $y' \in x \longrightarrow \cdot \longrightarrow x'$.

Consider a proper execution sequence $\vec{\sigma}$, where σ_i is specified by Figure 5; since $\chi_i(a_1) \neq \chi_i(a_2)$, $\vec{\sigma}$ does not satisfy the strong inter-operator fork timing assumption.

Taken together, the strong intra-operator fork and inter-operator fork timing assumptions are equivalent to the standard isochronicity assumption.

Definition 4.5. Let $\vec{\sigma}$ be a proper execution sequence. $\vec{\sigma}$ satisfies the *strong fork timing assumption* (SFTA) if and only if it satisfies the properties of Definitions 4.3–4.4.

Defined next is the notion of an adversary path [2], [3], a specific type of acknowledgment path beginning at one branch of a fork and looping around to the target of another branch of the same fork. Due to space limitations, the definition given next is considerably simplified from the version given in the extended version of this paper [10]. This allows for a much more compact proof in Section 5, while still maintaining the key ideas. The implications of this simplification are discussed in Section 5.7; for full details, see [10].

Definition 4.6. Let $\mathcal{O}_x, \mathcal{O}_u, \mathcal{O}_v$ be distinct gates such that $x \longrightarrow \cdot \longrightarrow u \neq \emptyset$ and $x \longrightarrow \cdot \longrightarrow v \neq \emptyset$. In addition, let $y \leftrightarrows_i x$ for some $y \in \cdot \longrightarrow x$. With respect to $y \leftrightarrows_i x$, an *adversary* is any acknowledgment path $x \leftrightarrows^+_{[i,j]} v \leftrightarrows^+_{[j,k]} x'$, with $x' \longrightarrow \cdot \longrightarrow u \neq \emptyset$, and where for all $y'' \in x \longrightarrow \cdot \longrightarrow u$ and $i < l \leq k$, $\chi_l(y'') \neq \chi_i(x)$.

Figure 6 completely exposes all inter-operator forks from Figure 1. For clarity, since the strong intra-operator fork timing assumption is assumed, intra-operator forks are not drawn. Consider a proper execution sequence $\vec{\sigma}$, where σ_i is specified by Figure 6. Now imagine that $\mathcal{O}_{a_1}$, the wire between the inverter and the C-element, transitions; *i.e.* $\chi_{i+1} = \chi_i[a_1 \mapsto \mathtt{F}]$. Next, the C-element transitions and $\chi_{i+2} = \chi_{i+1}[b \mapsto \mathtt{T}]$; there is now an acknowledgment path $a \leftrightarrows^+_{[i,i+1]} a_1 \leftrightarrows^+_{[i+1,i+2]} b$. This acknowledgment path is an adversary. Intuitively, this adversary path creates a potential instability at $\mathcal{O}_x$. For example, suppose that $\chi_{i+3} = \chi_{i+2}[b_1 \mapsto \mathtt{T}]$. This enables the NAND gate, $\mathcal{O}_x$, but the F at the output of the inverter, $\mathcal{O}_a$, can propagate to the a_2 input of the NAND gate at any step, disabling the NAND gate and generating an instability.

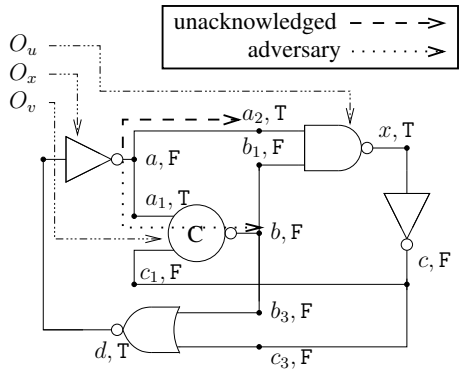

Figure 6. **Adversary path; at state** $\sigma_i = (\chi_i, \emptyset)$.

Definition 4.7. Let $\vec{\sigma}$ be a proper execution sequence. $\vec{\sigma}$ satisfies the *weak inter-operator fork timing assumption* if and only if $\vec{\sigma}$ contains no adversaries.

The weak inter-operator fork timing assumption yields a weaker assumption than SFTA. This weaker assumption is the *adversary path timing assumption* (APTA). Section 5 proves that the SFTA and APTA assumptions are equivalent with respect to the existence of hazards.

Definition 4.8. Let $\vec{\sigma}$ be a proper execution sequence. $\vec{\sigma}$ satisfies the *adversary path timing assumption* (APTA) if and only if it satisfies the properties of Definition 4.3 and Definition 4.7.

Next, following the classic definition [6], a circuit is defined as delay-insensitive if it is hazard-free under the assumption that wires, gates, and forks have arbitrary but finite delays.

Definition 4.9. Let $\mathcal{P}$ be a proper PRS. $\mathcal{P}$ is *delay-insensitive* (DI) with respect to reset state σ_1 if and only if for all proper execution sequences $\vec{\sigma}$ with reset state σ_1 and satisfying the properties of Definition 4.3, $\vec{\sigma}$ is stable and non-interfering.

Finally, this paper provides a definition for quasi-delay-insensitive and speed-independent circuits. In agreement with [11], a circuit is SI if it is hazard-free under the assumption that gates and wires can have arbitrary delays, as long as these delays are positive and finite, but all wire forks must transition at the same time, *i.e.* all sequences obey the strong intra-operator fork and strong inter-operator fork

timing assumptions. Similarly, a circuit is QDI if it is hazard-free (stable and non-interfering) under the assumption that gates and wires can have arbitrary (positive and finite) delays with all sequences obeying the strong-intra operator fork timing assumption, and with a subset of the forks, called isochronic forks, additionally obeying the strong inter-operator fork timing assumptions.

Definition 4.10. Let $\mathcal{P}$ be a proper PRS. $\mathcal{F}$ denotes the subset of operators of $\mathcal{P}$ that are wires.

Definition 4.11. Let $\mathcal{P}$ be a proper PRS, let $\mathcal{F}_1, \mathcal{F}_2$ partition $\mathcal{F}$, and assume that the constraints of Definitions 4.3 and 4.4 are satisfied by all forks in $\mathcal{F}_1$, *i.e.* they are isochronic, but that the forks in $\mathcal{F}_2$ are only required to satisfy the constraints of Definition 4.3.

(a) $\mathcal{P}$ is *speed-independent* (SI) w.r.t. to reset state σ_1 if and only if the set of proper execution sequences beginning with σ_1 are stable and non-interfering and $\mathcal{F}_2 = \emptyset$.

(b) $\mathcal{P}$ is *quasi-delay-insensitive* (QDI) w.r.t. to reset state σ_1 if and only if the set of proper execution sequences beginning with σ_1 are stable and non-interfering.

5. Equivalence of SFTA and APTA

Consider again Figure 5 and the operational definition of PRS computation from Section 3; there are a number of legal execution sequences from $\sigma_i = (\chi_i, \emptyset)$ that exhibit hazards. For example, there is an interference hazard when $\chi_{i+1} = \chi_i[b_1 \mapsto \mathtt{T}]$. The goal of timing assumptions, such as the strong intra-operator fork timing assumption (Definition 4.3), is to restrict the set of execution sequences so that hazards are excluded. Indeed, under the strong intra-operator fork timing assumption, the above execution step is impossible. Of course, different timing assumptions may exclude different execution sequences. Moreover, they may do so at different *costs*; *i.e.* some timing assumptions may be *weaker* (easier to satisfy with physical circuits) than others.

This section primarily addresses the set of execution sequences that are excluded under (a) the *strong fork timing assumption* (SFTA, Definition 4.5), and (b) the *adversary path timing assumption* (APTA, Definition 4.8). The goal is to show that whenever the strong fork timing assumption excludes *all* execution sequences exhibiting a hazard, then so does the adversary path timing assumption; and *vice versa*. Formally, the aim is to prove the following theorem:

Theorem 5.1. *Let $\mathcal{P}$ be a proper* PRS *and σ_1 a reset state. $\mathcal{P}, \sigma_1$ is stable and non-interfering with respect to the strong fork timing assumption (*SFTA*) if and only if $\mathcal{P}, \sigma_1$ is stable and non-interfering with respect to the adversary path timing assumption (*APTA*).*

With respect to the interference hazard from Figure 5 when $\chi_{i+1} = \chi_i[b_1 \mapsto \mathtt{T}]$, both SFTA and APTA exclude the execution sequence, because both entail the strong intra-operator fork timing assumption. (It is worth noting that the strong intra-operator fork assumption may not be strictly necessary for proper SI circuit operation, but relaxing it falls clearly in the realm of analog constraints and is orthogonal to the equivalence of SFTA and APTA.) As a second example, modify the initial state of Figure 5 so that $\sigma_i = (\chi_i[a_1, a_4 \mapsto \mathtt{T}, b \mapsto \mathtt{F}], \emptyset)$. Consider the instability hazard exposed through the execution sequence $\chi_{i+1} = \chi_i[a_1, a_4 \mapsto \mathtt{F}]$, $\chi_{i+2} = \chi_{i+1}[b \mapsto \mathtt{T}]$, $\chi_{i+3} = \chi_{i+2}[b_1, b_2 \mapsto \mathtt{T}]$, and $\chi_{i+4} = \chi_{i+3}[a_2, a_3 \mapsto \mathtt{F}]$. $\mathcal{O}_x$ is enabled at index $i+3$, but no longer enabled at index $i+4$ so that $I_{i+4} = \{x\}$. Both timing assumptions again reject this sequence, but this time for different reasons. The execution step from i to $i+1$ is rejected under SFTA because $\chi_{i+1}(a_1) \neq \chi_{i+1}(a_2)$. Under APTA this execution step is allowed, but what is not allowed is the sequence of acknowledgments $a \leftrightarrows_i a_1 \leftrightarrows_{i+1} b$.

The ($\Leftarrow$) direction of Theorem 5.1 is straightforward, and is given in Section 5.1. The ($\Rightarrow$) direction is substantially harder. Section 5.2 sketches the main idea of the proof of Theorem 5.1 ($\Rightarrow$) at a high level. Sections 5.3 – 5.6 develop the details, and Section 5.7 discusses several implications of the proof.

5.1. Theorem 5.1 ($\Leftarrow$)

Proof: Every SFTA execution sequence is also an APTA execution sequence. Toward a contradiction, assume there exists an execution sequence $\vec{\sigma}$ which is SFTA but not APTA. $\vec{\sigma}$ must contain an adversary path. Let $x \leftrightarrows^+_{[i,j]} v \leftrightarrows^+_{[j,k]} y'$ be as in Definition 4.6. By the definition, for some index l, $i < l < j$, and all z, z' with $z \in x \longrightarrow \cdot \longrightarrow v$ and $z' \in x \longrightarrow \cdot \longrightarrow u$, $\chi_l(z) \neq \chi_l(z')$. This contradicts the strong inter-operator fork timing assumption. □

5.2. Theorem 5.1 ($\Rightarrow$) Overview

The proof of Theorem 5.1 follows by contradiction: assuming that SFTA is stable and non-interfering, it is shown that the existence of a hazardous APTA sequence implies also a hazardous SFTA execution sequence, an obvious contradiction. The proof given in the following sections is constructive, and so given an APTA execution sequence $\vec{\sigma}$ with a hazard, it is shown how to construct an SFTA execution sequence that also has a hazard.

The construction crucially relies on the notions of *relaxation* (Definition 5.2) and *variant* execution sequence (Definition 5.4). Given an APTA execution sequence, $\vec{\sigma}$, a variant is a *modified execution sequence*, say $\vec{\sigma}'$, in which certain transitions on wires are either *forced* or *suppressed*. In Figure 5, assuming again a modified initial state $\sigma_i = (\chi_i[a_1, a_4 \mapsto \mathtt{T}, b \mapsto \mathtt{F}], \emptyset)$, consider going from $\sigma_i = (\chi_i, \emptyset)$ to $\chi_{i+1} = \chi_i[a_1, a_4 \mapsto \mathtt{F}]$. This execution step is APTA

but not SFTA because $\chi_{i+1}(a_1) \neq \chi_{i+1}(a_2)$. The condition where the branches of the $\mathcal{O}_a$ fork differ between gates $\mathcal{O}_x$ and $\mathcal{O}_b$ is called a *relaxation*, and the modifications that are made in a variant sequence are with respect to relaxations. One possible variant of the above execution step is to *force* a_2, a_3 to acknowledge a along with a_1, a_4, so that $\chi'_{i+1} = \chi_i[a_1, a_2, a_3, a_4 \mapsto \mathrm{F}]$; the second type of variant suppresses the acknowledgment of a on a_1, a_4, so that $\chi'_{i+1} = \chi_i$. Note that in both cases the modified execution sequence is SFTA.

A main insight of the proof is the identification of a gate, say $\mathcal{O}_u$, that is the *inherent origin of a hazard*. Moreover, the proof makes concrete the *forced/suppressed transitions needed to manifest this hazard at* $\mathcal{O}_u$. The gate is identified by considering the variant of $\vec{\sigma}$ in which *all relaxations are forced*, call this variant $\vec{\sigma}^+$, and is found at the smallest index, say j, where there is a gate, $\mathcal{O}_u$, such that $\chi_j(u) \neq \chi^+_j(u)$.

The details of Theorem 5.1 ($\Rightarrow$) are broken down as follows. Due to space limitations, certain details are given in [10]. Section 5.3 formally defines the notions of relaxation and variant. Section 5.3 also establishes (see Lemma 5.7) that *all variants are SFTA*. Section 5.4 isolates the hazard to a specific index and gate. Section 5.5 characterizes exactly how certain specific variants differ from the original APTA sequence. Finally, Section 5.6 demonstrates that the differences proved in the previous section are minor enough to yield a hazard in the SFTA variant when the APTA sequence has a hazard, which is finally proved in Section 5.6.

5.3. Relaxations and Variant Execution Sequences

The notion of *relaxation* encapsulates the idea that the first difference between the two timing assumptions manifests itself on the branches of a fork between two different gate operators. More specifically, using the weak inter-operator fork timing assumption, a signal may propagate to one gate at the end of a fork branch and not to another gate at the end of a different branch; while by definition, this is impossible under the strong inter-operator fork timing assumption.

Definition 5.2. Let $\vec{\sigma}$ be an APTA execution sequence. Associated to $\vec{\sigma}$ is a set of *relaxations*, $\mathcal{R}_{\vec{\sigma}}$, with

$$\mathcal{R}_{\vec{\sigma}} \subseteq V \times V \times \mathbb{N} \times \mathbb{N} \cup \{\infty\}$$

such that $(x, u, m, n) \in \mathcal{R}_{\vec{\sigma}}$ if and only if

- $\mathcal{O}_x, \mathcal{O}_u$ are gates, $m < n$, and $x \longrightarrow \cdot \longrightarrow u \neq \emptyset$.
- For some $y \in x \longrightarrow \cdot$; $x \leftrightarrows_m y$.
- For some $y \in x \longrightarrow \cdot$, $x \leftrightarrows_n y$; or $n = \infty$.
- For all $y \in x \longrightarrow \cdot$ and i such that $m < i < n$; $x \not\leftrightarrows_i y$.
- For all $y' \in x \longrightarrow \cdot \longrightarrow u$, $\chi_{m+1}(y') \neq \chi_m(x)$.

When $\vec{\sigma}$ is clear from context, $\mathcal{R}$ is used in place of $\mathcal{R}_{\vec{\sigma}}$. The essential idea behind constructing a *variant execution sequence* is to modify an APTA execution sequence *at relaxed forks*. There are two types of local modifications when a fork has a relaxation: the relaxed branches can be made to mimic the non-relaxed branches by *forcing transitions*; alternatively the non-relaxed branches can be made to mimic the relaxed branches by *suppressing transitions*. Conceptually, it is simpler to consider such modifications over a set of related relaxation points, a "*relaxation span*", rather than at every individual relaxation.

Definition 5.3. Let $\vec{\sigma}$ be an APTA execution sequence. The *relaxation span set*,

$$\mathcal{S}_{\vec{\sigma}} \subseteq V \times \mathbb{N} \times \mathbb{N} \cup \{\infty\}$$

is defined such that for every maximal sequence of relaxations

$$(x, u_1, m, i_1)\ (x, u_2, i_1, i_2)\ \cdots\ (x, u_k, i_{k-1}, n)$$

with $\chi_i(x) = \chi_{m-1}(x)$ for all $m \leq i \leq i_{k-1}$; $(x, m, n) \in \mathcal{S}_{\vec{\sigma}}$.

In the above definition, "maximal" means that there is no longer sequence which includes the given one. Consider an (APTA) execution sequence $\vec{\sigma}$. The formal definition of variant attempts to mimic $\vec{\sigma}$ as closely as possible *except with the relaxation spans*. For a span $(x, m, n) \in \mathcal{S}$, the branches $y \in x \longrightarrow \cdot$ of the fork from $\mathcal{O}_x$ are either all *forced*, or all *suppressed*. Since every relaxation is part of a span, and across a span all branches of a fork are treated equally, a variant will always be SFTA. This is proved formally in Lemma 5.7.

The set $\mathcal{S}^+$ in the definition of variant corresponds to spans which are *forced*. The set $\mathcal{S}^-$ corresponds to spans which are *suppressed*. The definition is broken up into pieces to facilitate explanation of the construction.

Definition 5.4. Let $\vec{\sigma}$ be an APTA execution sequence and let $\mathcal{S}^+, \mathcal{S}^-$ partition $\mathcal{S}$. The *variant* of $\vec{\sigma}$ with respect to $\mathcal{S}^+, \mathcal{S}^-$ is the execution sequence, say $\vec{\sigma}'$, such that $\sigma'_1 = \sigma_1$, and ...(continued below)

For gates, $\vec{\sigma}'$ should *always* mimic $\vec{\sigma}$ if it can, and otherwise a default action should be taken. The default action forces the previous value of x to persist across the execution step.

Definition 5.5. Let $\mathcal{P}$ be a proper PRS and $\chi, \chi' : V \longrightarrow \mathbb{B}$. For any operator $\mathcal{O}_x$, χ, χ' *agree* on $\mathcal{O}_x$, $\chi(\mathcal{O}_x) \Leftrightarrow \chi'(\mathcal{O}_x)$, if and only if they give the same interpretation with respect to the predicates of Definition 3.3.

(Definition 5.4 Cont., Gates). ...for $i+1 > 1$ and x such that $\mathcal{O}_x$ is a gate:

$$\chi'_{i+1}(x) = \chi_{i+1}(x) \quad \text{if } \chi'_i(\mathcal{O}_x) \Leftrightarrow \chi_i(\mathcal{O}_x) \text{ and } y \leftrightarrows_i x \text{ for some } y \tag{1a}$$

$$\chi'_{i+1}(x) = \chi'_i(x) \quad \text{otherwise} \tag{1b}$$

The same basic strategy employed for gates is also used for wires, *except* across a span.

(Definition 5.4 Cont., Wires). ...for $i+1 > 1$ and y such that $\mathcal{O}_y$ is a gate with $y \in x \longrightarrow \cdot$:

$$\chi'_{i+1}(y) = \chi_m(x) \quad \text{if } \chi'_m(\mathcal{O}_y) \Leftrightarrow \chi_m(\mathcal{O}_y) \text{ and there exists a } (x,m,n) \in \mathcal{S}^+ \text{ with } m \leq i < n \tag{2a}$$

$$\chi'_{i+1}(y) = \chi'_m(y) \quad \text{if } \chi'_m(\mathcal{O}_y) \Leftrightarrow \chi_m(\mathcal{O}_y) \text{ and there exists a } (x,m,n) \in \mathcal{S}^- \text{ with } m \leq i < n \tag{2b}$$

$$\chi'_{i+1}(y) = \chi_i(x) \quad \text{if } \chi'_i(\mathcal{O}_y) \Leftrightarrow \chi_i(\mathcal{O}_y), \text{ there is } \textit{no } (x,m,n) \in \mathcal{S}^+ \cup \mathcal{S}^- \text{ with } m \leq i < n, \text{ and } x \leftrightarrows_i y' \text{ for some } y' \in x \longrightarrow \cdot \tag{2c}$$

$$\chi'_{i+1}(y) = \chi'_i(y) \quad \text{otherwise} \tag{2d}$$

It is straightforward to show that Definition 5.4 yields a well-defined execution sequence. Finally, Lemma 5.7 demonstrates that all variant execution sequences are SFTA.

Lemma 5.6. *Let $\vec{\sigma}$ be an* APTA *execution sequence and $\vec{\sigma}'$ a variant of $\vec{\sigma}$. $\vec{\sigma}'$ is a proper execution sequence with the same reset state as $\vec{\sigma}$.*

Proof: Straightforward; see [10]. □

Lemma 5.7. *Let $\vec{\sigma}$ be an* APTA *execution sequence and $\vec{\sigma}'$ a variant of $\vec{\sigma}$. $\vec{\sigma}'$ is* SFTA.

Proof: By induction. It is sufficient to show that for all gates $\mathcal{O}_x$ and every index i, if $y, y' \in x \longrightarrow \cdot$, then $\chi'_i(y) = \chi'_i(y')$. At $i = 1$, $\sigma'_1 = \sigma_1$ and the result follows from the definition of a proper execution sequence reset state. Assume the result up to i; it must be shown to extend to $i+1$.

Toward a contradiction, suppose $\chi'_{i+1}(y) \neq \chi'_{i+1}(y')$ for some such y, y' as above. It is easy to show that for any of the cases, if $\chi'_{i+1}(y)$ is defined by that case, then so is $\chi'_{i+1}(y')$. Clearly, both $\chi'_{i+1}(y)$ and $\chi'_{i+1}(y')$ cannot be defined by case (2a) or case (2c) (this would force $\chi'_{i+1}(y) = \chi'_{i+1}(y') = \chi_m(x)$ or $\chi'_{i+1}(y) = \chi'_{i+1}(y') = \chi_i(x)$, respectively); similarly, $\chi'_{i+1}(y), \chi'_{i+1}(y')$ cannot both be defined by case (2b) or both be defined by case (2d) (by the induction hypothesis). □

5.4. Isolating the Hazard

Let $\vec{\omega}$ be some unstable or interfering APTA execution sequence. This execution sequence is carried through the remainder of the proof of Theorem 5.1 ($\Rightarrow$), and is used to distinguish from $\vec{\sigma}$, which is used more generally. From $\vec{\omega}$ a "refined" APTA sequence is generated, $\vec{\omega}'$, and it is proved that a specific *variant* of $\vec{\omega}'$, $\vec{\omega}'^-$, also contains a hazard. This implies a contradiction because all variants are SFTA.

$\vec{\omega}'$ is constructed so as to isolate the hazard to an index j and specific gate $\mathcal{O}_u$. The construction is notable because the differences between $\vec{\omega}'$ and $\vec{\omega}'^-$ are extremely limited. The exact differences are given by Lemma 5.11. This allows for a relatively straightforward comparison of $\vec{\omega}'$ and $\vec{\omega}'^-$ showing that the variant sequence contains a hazard. Index j and gate $\mathcal{O}_u$ are found by comparing $\vec{\omega}'$ with $\vec{\omega}'^+$, the variant of $\vec{\omega}'$ where all relaxation spans are forced.

Definition 5.8. Let $\vec{\sigma}$ be an APTA execution sequence. $\vec{\sigma}^+$ denotes the variant of $\vec{\sigma}$ with respect to $\mathcal{S}, \emptyset$.

Refinement. Consider $\vec{\omega}$ and $\vec{\omega}^+$. Let j be the smallest index such that either $I_j \neq \emptyset$ or for some gate $\mathcal{O}_u$, $\chi_{j+1}(u) \neq \chi^+_{j+1}(u)$. Refine $\vec{\omega}$ to the execution sequence $\vec{\omega}'$ as follows ...(continued below)

The refinement branches based on the two conditions, *i.e.* $I_j \neq \emptyset$ or not. The details of the first case are omitted for space (see [10]) but are quite similar to when $I_j = \emptyset$. The second case is the key idea developed for the proof. $\chi_{j+1}(u) \neq \chi^+_{j+1}(u)$ indicates the potential for an instability hazard at $\mathcal{O}_u$. It will be shown that, essentially, if all of the relaxed forks leading to $\mathcal{O}_u$ at index j are forced to transition, then $\mathcal{O}_u$ becomes disabled. The remainder of the Refinement and proofs below deal with the details of demonstrating this result formally.

(Refinement Cont., $(I_j = \emptyset)$). Let $\mathcal{Z} \subseteq \mathcal{S}$ be such that for all $(x,m,n) \in \mathcal{S}$, $(x,m,n) \in \mathcal{Z}$ if and only if there is a $(x,u,k,l) \in \mathcal{R}$ with $m \leq k < j \leq l \leq n$. For all $i \leq j$ and $x \in V$

$$\chi'_i(x) = \chi_m(x) \quad \text{if there exists a } (x,m,n) \in \mathcal{Z} \text{ with } m < i \leq n.$$

$$\chi'_i(x) = \chi_i(x) \quad \text{otherwise}$$

for $i = j + 1$

$$\chi'_{j+1}(x) = \chi_m(z) \quad \text{if there exists a } (z,m,n) \in \mathcal{Z} \text{ with } z \longrightarrow x.$$

$$\chi'_{j+1}(x) = \chi'_j(x) \quad \text{otherwise}$$

and for all $i > j + 1$, $\omega'_i = \omega'_{j+1}$.

Lemma 5.9. *Let $\vec{\omega}$ and $\vec{\omega}'$ be as in the refinement; $\vec{\omega}'$ is an* APTA *execution sequence with the same reset state as w.*

The main intuition as to why Lemma 5.9 is true comes from the fact that for $i \leq j$, $\vec{\omega}$ and $\vec{\omega}'$ differ at some gate $\mathcal{O}_x$ if and only if $(x,m,n) \in \mathcal{Z}$, where $m < j \leq n$. By Definitions 5.2 and 5.3, for all $y \in x \longrightarrow \cdot$, $m < k < n$, $x \not\leftrightarrows_k y$. That is, even if $\mathcal{O}_x$ transitions at some state, say ω_k, $m < k < n$, no wire could have observed this transition prior to ω_n. As such, x can clearly be held at its initial state in ω_m until ω_n.

Proof: See [10]. □

5.5. Constructing the Hazardous SFTA Sequence

The SFTA variant of $\vec{\omega}'$ that will be shown to contain a hazard is defined so that every relaxation span $(x, m, n) \in \mathcal{Z}$ is suppressed, while every other relaxation span is forced. This execution sequence is denoted $\vec{\omega}'^{-}$. Every difference between $\chi'_i(x)$ and $\chi'^{-}_i(x)$ gets accounted for in Lemma 5.11 below. Unless x is the relaxed branch of a fork from a span $(x', m, n) \in \mathcal{S} \setminus \mathcal{Z}$, then the lemma essentially shows that $\chi'_i(x)$ is on an acknowledgment path from a suppressed $(x', m, n) \in \mathcal{Z}$.

Definition 5.10. Let $\vec{\sigma}$ be an APTA execution sequence and $\mathcal{Z} \subseteq \mathcal{S}$. $\vec{\sigma}^{-}$ denotes the variant of $\vec{\sigma}$ with respect to $\mathcal{S} \setminus \mathcal{Z}$, $\mathcal{Z}$.

Lemma 5.11. *Let $\vec{\omega}'$, j, $\mathcal{O}_u$, and $\mathcal{Z}$ be as in the Refinement. With respect to $\vec{\omega}'^{-}$, for all $i \leq j$ and x, if $\chi'_i(x) \neq \chi'^{-}_i(x)$ then either*

(a) *there is a $(x', m, n) \in \mathcal{S} \setminus \mathcal{Z}$ with $x \in x' \longrightarrow \cdot$ and relaxed at $m < i \leq n$,*
(b) *there is a $(x', m, n) \in \mathcal{Z}$ such that $x' \leftrightarrows^{+}_{[m,i]} x$ in $\vec{\omega}'$.*

Proof: see [10]. □

5.6. Theorem 5.1 ($\Rightarrow$)

Proof:

($I_j = \emptyset$). Let $y \in \cdot \longrightarrow u$ be such that $\chi'^{-}_j(y) \neq \chi'_j(y)$. By Lemma 5.11, either (a) there is a $(x, m, n) \in \mathcal{S} \setminus \mathcal{Z}$ and $x \longrightarrow \cdot \longrightarrow u$ is relaxed at $m < j \leq n$, or (b) there is a $(x, m, n) \in \mathcal{Z}$ and $x \leftrightarrows^{+}_{[m,j]} y$ in $\vec{\omega}'$. Case (a) is impossible by the construction of the $\mathcal{Z}$ set, and case (b) is impossible by the definition of adversary path ($x \longrightarrow \cdot \longrightarrow u$ is relaxed at $m < j \leq n$, yet there is an acknowledgment from x at m that leads back to $\mathcal{O}_u$ at index j). Therefore, for all $y \in \cdot \longrightarrow u$, $\chi'^{-}_j(y) = \chi'_j(y)$. By similar reasoning, $\chi'^{-}_j(u) = \chi'_j(u) = \chi_j(u)$. It remains to be shown that $\chi'_{j+1}(y) = \chi^{+}_j(y)$ for all $y \in \cdot \longrightarrow u$ (this implies an instability in $\vec{\omega}'^{-}$). A simple corollary to Lemma 5.11 establishes that $\chi^{+}_j(y) \neq \chi'_j(y)$ exactly on those $y \in x \longrightarrow \cdot \longrightarrow u$ that are relaxed at j. By the construction of the refinement and case (2c) of the definition of variant, these are exactly the y that change in the execution step from χ'^{-}_j to χ'^{-}_{j+1}. Therefore, for all $y \in \cdot \longrightarrow u$, $\chi'^{-}_{j+1}(y) = \chi^{+}_j(y)$ and $\mathcal{O}_u$ gets disabled, an instability and a contradiction that SFTA is stable and non-interfering.

($I_j \neq \emptyset$). see [10]. □

5.7. Discussion

From the proof of Theorem 5.1, it is clear that for *certain* forks, the adversary path timing assumption is in some sense the *weakest* timing assumption equivalent to the strong fork timing assumption; no path longer than an adversary path can yield a timing assumption that is equivalent to APTA or SFTA with respect to the existence of hazards. However, suppose that the forks within a proper PRS are partitioned into isochronic and non-isochronic forks, *i.e.* into the sets $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively (from Definitions 4.10 and 4.11). Since no timing assumption whatsoever need be applied to the forks in $\mathcal{F}_2$, the assumption of APTA (as defined in this paper) may disallow the final gate in an $\mathcal{F}_2$ adversary path from transitioning even though the resulting transition would never generate a hazard.

Partitioning the forks as described, and applying APTA exclusively to the forks in the $\mathcal{F}_1$ set can also unnecessarily prevent the final gate in an adversary path from transitioning. Moreover, this can occur for a number of reasons, *e.g.* some fork in $\mathcal{F}_1$ is isochronic for up-going transitions but not for down-going transitions or *vice versa*. To properly address this issue, [10] defines an extended version of adversary path; however, the extended definition is considerably more complicated both intuitively and technically, and the corresponding equivalence proof is therefore more involved.

6. Related Work

The importance of understanding timing assumptions in asynchronous circuits is well-known [6], [12]. Furthermore, it is recognized that strict *isochronicity* (SFTA) is both difficult to satisfy [13] and unnecessary for hazard-free operation. The adversary path is described directly in [3], and similar timing assumptions are described in [14], [15]. These works provide important intuition as to why relaxing the strong fork timing assumption to the adversary path assumption is sufficient for ensuring hazard-free operation. However, they do so without a formal framework, and hence without proof of correctness. Moreover, they do not provide any intuition as to why the adversary path timing assumption is the *weakest* timing assumption that is both necessary and sufficient for correct operation of SI circuits.

Other works have generated useful extensions of SI circuits that relax isochronic forks, *e.g.* the extended isochronic fork from [16]. The extended isochronic fork allows for additional gates to be placed on the unacknowledged branch of a fork and can yield more compact circuits. It seems clear that the adversary path assumption naturally extends to this assumption; although a formal proof establishing this is not given. Similarly, the timing constraint on orphans in NULL convention logic is almost certainly a specific variant of an adversary path; but, again, this is not formally established in this paper. However, by providing a formalization at the level of switching networks, the current work could be extended to investigate such issues further.

7. Conclusion

This paper presents a complete formalization of the notion of production rule sets, a well-known asynchronous computation system. Using this system, a number of fork-related timing assumptions are also formalized, including the adversary path assumption, and these formalizations are employed in order to characterize several important asynchronous logic frameworks. Finally, it is proved that the adversary path timing assumption is both a necessary and a sufficient condition for correct operation of speed-independent circuits and various extensions of SI circuits.

However, the model of computation presented, like all models, has limitations. First, it does not provide syntactic or semantic support for pass-transistors. Second, it does not directly include the transistors required to physically reset a PRS. Third, wires are assumed to be perfect. Finally, the model does not support interfering state-holding circuits such as cross-coupled inverters. The first two limitations are relatively minor; they have been excluded for clarity. The last two limitations require considerable effort to remedy without excessively encumbering the specification of the system, and are therefore left as future work.

Acknowledgment

Acknowledgment is due to Chris J. Myers, Piyush Prakash, and to the members of our research groups for their excellent comments and insights. The research described in this paper is in part supported by a grant from the National Science Foundation.

References

[1] A. J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, no. 4, 1986.

[2] A. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proceedings of the IEEE*, vol. 94, no. 6, 2006.

[3] A. J. Martin and P. Prakash, "Asynchronous nano-electronics: Preliminary investigation," in *Asynchronous Circuits and Systems, 2008. ASYNC '08. 14th IEEE International Symposium on*, 2008.

[4] D. E. Muller, W. Bartky, and S, "A theory of asynchronous circuits," in *Laboratory of Harvard University, Vole 29, Part I, Harvard University Press*, 1959.

[5] R. E. Miller, *Switching Theory, Volume II: Sequentical Circuits and Machines*. John Wiley & Sons, Inc., 1965.

[6] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *AUSCRYPT '90: Proceedings of the sixth MIT conference on Advanced research in VLSI*. MIT Press, 1990.

[7] D. B. Armstrong, A. D. Friedman, and P. R. Menon, "Design of asynchronous circuits assuming unbounded gate delays," *IEEE Trans. Comput.*, vol. 18, no. 12, 1969.

[8] K. Papadantonakis, "Design rules for non-atomic implementation of PRS," California Institute of Technology, Tech. Rep. CaltechCSTR:2005.001, 2005.

[9] A. De Gloria, P. Faraboschi, and M. Olivieri, "Design and characterization of a standard cell set for delay insensitive VLSI design," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 41, no. 6, 1994.

[10] S. Keller, M. Katelman, and A. J. Martin, "A necessary and sufficient timing assumption for speed-independent circuits (extended version)," California Institute of Technology, Tech. Rep. CaltechCSTR:2009.001, 2009. [Online]. Available: http://resolver.caltech.edu/CaltechCSTR:2009.001

[11] P. Beerel, J. Burch, and T.-Y. Meng, "Sufficient conditions for correct gate-level speed-independent circuits," in *Advanced Research in Asynchronous Circuits and Systems, 1994., Proceedings of the International Symposium on*, 1994.

[12] K. Stevens, R. Ginosar, and S. Rotem, "Relative timing [asynchronous design]," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 1, 2003.

[13] K. van Berkel, "Beware the isochronic fork," *Integr. VLSI J.*, vol. 13, no. 2, 1992.

[14] K. Fant, *Logically Determined Design*. John Wiley & Sons, Inc., 2005.

[15] N. Sretasereekul and T. Nanya, "Eliminating isochronic-fork constraints in quasi-delay-insensitive circuits," in *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, 2001.

[16] K. van Berkel, F. Huberts, and A. Peeters, "Stretching quasi delay insensitivity by means of extended isochronic forks," *Asynchronous Design Methodologies, 1995. Proceedings., Second Working Conference on*, 1995.