Analyzing Isochronic Forks with Potential Causality

Rajit Manohar
Cornell NYC Tech
New York, NY 10011, USA
rajit@csl.cornell.edu

Yoram Moses†
Technion-Israel Institute of Technology
Haifa 32000, Israel
moses@ee.technion.ac.il

***Abstract*—Asynchrony and concurrency are fundamental notions in the fields of asynchronous circuits as well as distributed systems. This paper treats asynchronous circuits as a special class of distributed systems. We adapt the distributed systems notion of potential causality to asynchronous circuits, and use it to provide a formal proof of the precise nature of the isochronic fork timing assumption in quasi delay-insensitive (QDI) circuits. Our proofs provide a transparent analysis that provides better intuition regarding the operation of QDI circuits. We build on our theory to rigorously establish several "folk theorems" about identifying isochronic forks in QDI circuits.**

I. Introduction

There are a variety of approaches to the design of asynchronous circuits, each using a different set of assumptions about the timing properties of gates. Speed-independent (SI) circuits are those that operate correctly in the presence of arbitrary gate delays, but wire delays are assumed to be negligible. Quasi delay-insensitive (QDI) also assume arbitrary gate delays, but the delays of some wires that fork to multiple gates—known as *isochronic* forks—are assumed to be small [10]. Purely delay-insensitive circuits (DI) operate correctly in the presence of arbitrary gate and wire delays [10]. There are many other approaches to timing in asynchronous circuits, these three classes are the most conservative in what they assume about the underlying circuit technology.

A natural question that arises is the following: what is the expressive power of each class of circuits? In a well-known paper, Martin argued that the class of DI circuits is quite limited [10], providing a justification for considering timing in asynchronous circuits beyond considerations of performance, energy, or area. The class of QDI circuit is "in-between" DI and SI in terms of the assumptions made on timing,[1] because assuming negligible wire delays (the SI assumption) automatically implies that the isochronic fork timing assumption would be satisfied under the QDI model. Follow-on work has shown that QDI circuits are Turing-complete [8]. Therefore, the isochronic fork assumption is strong enough to enhance the computational power of asynchronous circuits so that any computable function (modulo finite memory) can be designed under the QDI model.

Since the isochronic fork is so fundamental, it is important to have a precise, formal characterization of the nature of the isochronic fork timing assumption. Suppose a gate output x forks to two other gates, and label the other end of the two end-points x_1 and x_2 as shown in Fig. 1. Furthermore, assume that this is an isochronic fork. The original description of the isochronic fork states that "we have to assume that the difference between the delays in the branches of the fork is negligible compared to the delays in the gates." [10]. Other descriptions are less conservative. In an example, [8] examines the scenario where a signal transition $x_1\uparrow$ causes $y\uparrow$, and states "we assume that, when transition $x_1\uparrow$ has been acknowledged by transition $y\uparrow$, transition $x_2\uparrow$ is also completed." In other words, as long as x_2 has changed by the time y changes, the isochronic fork assumption has been satisfied.

The most recent description that formalizes the nature of the isochronic fork timing assumption instead examines the impact of an *adversary path* [6]. In Fig. 1, an adversary path (shown by a broken line) corresponds to a sequence of gates whose delay competes with the wire from x to x_2. To establish the existence of such a path, [6] introduces a number of new definitions including a new execution model that tracks switching hazards, syntactic properties of Boolean expressions that correspond to pull-up and pull-down switching networks, relaxations of executions, and other technical properties. Even with this machinery, the proof of the main theorem is only sketched in [6], with the details to be provided in a technical report. To date, this report has not been completed [5]. Thus, the proof of the main result establishing the nature of the isochronic fork timing assumption is unavailable.

The distributed systems field studies the behavior of concurrent systems that consist of asynchronous processes that communicate via channels—which is identical to the high level description of asynchronous circuits using a language such as communicating hardware processes (CHP) [9]. Therefore, most of the results in the distributed systems literature translate to asynchronous circuits described at the CHP level of abstraction. In this paper, we adapt a key concept in the distributed systems community—*potential causality*—to asynchronous circuits described as a collection of gates and wires, rather than processes and channels.

Our gate-level definition of potential causality is used to formally establish a number of key properties of isochronic and non-isochronic branches in QDI circuits. Some of these are "folk theorems" that are intuitively understood by practitioners of QDI circuits, but they have not previously been

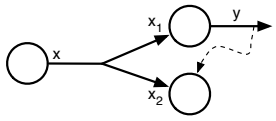

Fig. 1. A wire fork from the output of a gate x to the input of two other gates. The two end-points of the fork are labelled x_1 and x_2.

† Yoram Moses is the Israel Pollak academic chair at the Technion. This work was supported in part by the ISF grant 1520/11, and by a Ruch grant from the Jacobs Institute at Cornell Tech.

[1] Note that the QDI and SI assumptions define the same set of circuits.

rigorously established using complete formal proofs. The main contributions of this paper are:

- We introduce the notion of potential causality to the gates and circuits framework, adapting it from the distributed systems literature. Moreover, we formalize its role in circuit computations by proving the *past* theorem (Theorem 1).
- We prove the *firing loop* theorem (Theorem 2), which shows that feedback loops going through every output wire must exist between successive firings of a gate.
- Our main result is a complete and concise proof of the *adversarial firing chain* theorem (Theorem 3), which precisely characterizes the nature of the isochronic fork timing assumption.
- Finally, we illustrate how our results can be used to identify isochronic branches in a circuit from its set of production rules.

By providing crisp definitions and concise proofs of these essential aspects of asynchronous circuits, this paper contributes to the goal of providing a sound and effective mathematical theory for the field of asynchronous and self-timed systems. Moreover, we believe that our proofs, while technical, provide insight into the nature of isochronic forks in asynchronous circuits.

We begin by defining the execution model for asynchronous circuits, and review some of the standard terminology used in the literature (Section II). We also introduce a few key concepts from distributed systems that are used to reason about causality (Section II). Our main theorem about causality provides a formal way to show that the state of a gate at a point in time is only dependent on other gates in the past that have a causal connection to it (Section III). We then introduce the concepts of stability and non-interference of gates, and examine the impact of assuming that a gate is stable and non-interefering. Our main contribution is a complete proof of the adversarial firing chain theorem (Section IV). We then discuss related work in the asynchronous design community and in the distributed systems literature (Section V).

II. Definitions and Model

We assume the basic terminology of asynchronous circuits, production rules (PR) and effective firings as in [10]. We repeat the definitions for completeness.

A ***production rule*** (PR) over a set V of binary variables has the form $B \mapsto z\uparrow$ or $B \mapsto z\downarrow$, where $z \in V$ is a variable, and B is a propositional formula over a subset of the variables of V. A ***gate*** is a pair of production rules $B_u \mapsto z\uparrow$ and $B_d \mapsto z\downarrow$ for the same variable z. If y is a variable used in B_u or B_d, then we say that y is an ***input*** to the gate for variable z. A circuit over a set of variables V is a set of $|V|$ gates, one per variable $z \in V$.

Consider a circuit A over a set V_A of variables. A ***configuration*** of A is an assignment $c : V_A \to \{0, 1\}$ of binary values to the variables of V_A. Thus, if $V_A = \{x_1, \ldots, x_n\}$ then we can think of a *configuration* of A as an assignment $\vec{x} = \langle b_1, \ldots, b_n \rangle$ of binary values so that $x_i = b_i$ for all $1 \leq i \leq n$.

A production rule with guard B is *enabled* in a configuration c if its guard is true there. (This is denoted by $c \models B$.) The execution of an enabled production rule for variable x is called a *firing*. The firing causes the right hand side of the PR to be executed, so that the output of the PR is set to true (i.e., 1) or false (i.e., 0) depending on the right hand side of the PR. If this leaves the state (i.e., value) of x unchanged it is called a ***vacuous firing***, and if it causes a state change it is an ***effective firing***.

A. Computations

A ***computation*** of the circuit A is an infinite sequence $s : \mathbb{N} \to \mathcal{C}$ of configurations, such that $s(m+1)$ is obtained from $s(m)$ by firing zero or more PRs that are enabled at $s(m)$, for all $m \geq 0$. (A computation is finite in this setting if there exists a finite N_0 such that $s(m)$ is constant for all $m > N_0$.) We can also define notions of fairness of computations, e.g., if a PR is enabled continuously then it fires eventually, but that will not play a role in the particular analysis that we perform in this paper. The value of a variable x at time t is denoted by $s_x(t)$. We say that ***the value of*** x ***changes at time*** t ***in*** s if $s_x(t+1) \neq s_x(t)$. In natural scenarios there may be a set of legal initial configurations for the circuit. Computations will only be allowed to start in one of these. A general configuration is then *legal* if it is reachable in a computation that starts in a legal initial state.

Note that our definition of computations allows steps in which no variable values change, i.e., there can be times t such that $s(t+1) = s(t)$. In this case we say that there is a ***skip step*** at time t in s. Given a general execution s, we define its ***stuttering-free*** variant, denoted by $\underline{s}$, to be the execution obtained from s by removing all skip steps. In the Appendix we give a formal definition of $\underline{s}$ and show that if s is a computation of circuit A, then so is $\underline{s}$. It is clear that if s is a computation of A, the introduction or removal of a finite number of skip steps to or from s results in another valid computation of A.

Configurations $s(t)$ in a computation s are indexed by an integer t that plays the role of an external notion of time. (We also use m as an index for time.) But the circuit elements do not have access to this index, and it does not affect their operation. Reasoning from the outside, we will be interested in distinguishing the state of a variable x_i at different times m. This state, denoted by the pair $\langle x_i, m \rangle$, is called a ***node***.

B. Potential Causality

We adapt Lamport [7] and define potential causality as a binary relation over nodes on a circuit computation. Given a computation s, we write $\langle y, m \rangle \hookrightarrow_s \langle z, m+1 \rangle$ if a PR with output z performs an effective firing at $s(m)$, and the guard of this PR contains either y or $\neg y$.

Example. Consider the example circuit in Fig. 2. The initial state of the circuit is 0011, where the bit vector represents the value $xyab$. The first firing in the circuit is $y\uparrow$ at time 0, and there are two possible next firings $a\downarrow$ and $b\downarrow$. An example execution of this circuit is the sequence of actions $y\uparrow; a\downarrow; b\downarrow; x\uparrow; y\downarrow; \ldots$, which is represented by the computation $s = 0011; 0111; 0101; 0100; 1100; 1000; \ldots$ Finally, in this computation s we have that $\langle y, 1 \rangle \hookrightarrow_s \langle a, 2 \rangle$ and $\langle y, 2 \rangle \hookrightarrow_s \langle b, 3 \rangle$. □

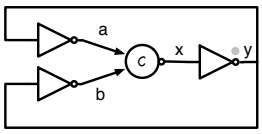

Fig. 2. Simple circuit example to illustrate concepts. Initially $a = b = 1$, $x = 0$, and $y = 0$. The grey dot next to y indicates that the first effective firing in this circuit is a change in y.

For each computation s, we define a partial order $\preceq_s$ over variable-time nodes called *potential causality* in s, to be the unique minimal relation satisfying the following three conditions:

Locality:	$\langle y,t\rangle \preceq_s \langle y,t'\rangle$	if $t \le t'$;
Successor:	$\langle y,t\rangle \preceq_s \langle z,t+1\rangle$	if $\langle y,t\rangle \hookrightarrow_s \langle z,t+1\rangle$;
Transitivity:	$\langle y,t\rangle \preceq_s \langle z,t'\rangle$	if, for some $\langle x,m\rangle$, both $\langle y,t\rangle \preceq_s \langle x,m\rangle$ and $\langle x,m\rangle \preceq_s \langle z,t'\rangle$.

Example. Continuing our example from Fig. 2, we can assert that $\langle y,1\rangle \preceq_s \langle x,4\rangle$. □

C. Firing Chains and Past

When $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ where $y \neq z$, the definition of potential causality implies that there must be a sequence of variable changes that are linked by the successor relationship. This is captured by the notion of *firing chains*.

Definition 1: We say that ***there is a chain of firings from*** $\langle y,t\rangle$ ***to*** $\langle z,t'\rangle$ ***in the computation*** s if there is a sequence of variables $x_1,\ldots,x_k = z$ and a sequence of monotonically increasing times $t_1,\ldots,t_k$ with $t \le t_1$ and $t_k < t'$, such that $\langle y,t_1\rangle \hookrightarrow_s \langle x_1,t_1+1\rangle$ and such that $\langle x_{i-1},t_i\rangle \hookrightarrow_s \langle x_i,t_i+1\rangle$ holds for all $2 \le i \le k$.

Example. Continuing our example from Fig. 2, there is a chain of firings from $\langle y,1\rangle$ to $\langle y,5\rangle$ established by the sequence of variables a, x, and y. □

A rather straightforward consequence of the definition of $\preceq_s$ is captured by

Lemma 1: Let $y \neq z$. Then $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ iff both $t < t'$ and there is a chain of firings from $\langle y,t\rangle$ to $\langle z,t'\rangle$ in s.

Proof: If $t < t'$ and there is a chain of firings from $\langle y,t\rangle$ to $\langle z,t'\rangle$ in s, then using Definition 1 and the definition of potential causality immediately gives us that $\langle y,t\rangle \preceq_s \langle z,t'\rangle$. We now prove the other direction.

Assume that $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ and $y \neq z$. We prove the claim by induction on the construction of $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ using Locality, Successor, and Transitivity.

- $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ cannot be obtained by Locality since $y \neq z$ and Locality relates only nodes with the same variable.
- If $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ is obtained by the Successor clause, then $t' = t+1$ and z changes value at time t. Thus, $t' > t$ and the sequence with $k = 1$, $x_1 = z$ and $t_1 = t$ satisfies the condition establishing that there is a chain of firings from $\langle y,t\rangle$ to $\langle z,t'\rangle$ in s.
- Assume that $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ is obtained by Transitivity, based on $\langle y,t\rangle \preceq_s \langle x,m\rangle$ and $\langle x,m\rangle \preceq_s \langle z,t'\rangle$. Moreover, assume inductively that the claim holds for the two derivations being combined. Since $y \neq z$ it follows that at least one of $y \neq x$ or $x \neq z$ holds. If $y \neq x$ then $t < m \le t'$ and if $x \neq z$ then $t \le m < t'$, so we immediately obtain that $t < t'$. It is easy to check that if only one of the two inequalities holds, then the sequence $x_1,\ldots,x_k$ and times $t_1,\ldots,t_k$ guaranteed for that relation by the inductive hypothesis satisfy the conditions of the claim. If $y \neq x \neq z$, then by the inductive hypothesis there are sequences $x_1,\ldots,x_k$ and $t_1,\ldots,t_k$ establishing a chain of firings from $\langle y,t\rangle$ to $\langle x,m\rangle$ in s and sequences $x'_1,\ldots,x'_{k'}$ and $t'_1,\ldots,t'_{k'}$ establishing a chain of firings from $\langle x,m\rangle$ to $\langle z,t'\rangle$ in s. The concatenated sequences of variables $x_1,\ldots,x_k,x'_1,\ldots,x'_{k'}$ and of times $t_1,\ldots,t_k,t'_1,\ldots,t'_{k'}$ satisfy the conditions of the claim.

■

Another useful notion in our model is the notion of the *past* of a node, which consists of the set of nodes that are related to it by the potential causality relation. More generally, we define the past of a set of nodes as follows.

Definition 2 (Past): Given a computation s and a set T of variable-time nodes, we define:

$$\mathtt{past}_s(T) = \bigcup_{\langle y',m'\rangle \in T} \{\langle x,m\rangle : \langle x,m\rangle \preceq_s \langle y',m'\rangle\} \ .$$

Example. Continuing our example from Fig. 2,

$$\begin{aligned}\mathtt{past}_s(\{\langle y,5\rangle\}) = \ & \{\langle y,5\rangle,\langle y,4\rangle,\langle y,3\rangle,\langle y,2\rangle,\langle y,1\rangle,\langle y,0\rangle,\\ & \langle a,2\rangle,\langle a,1\rangle,\langle a,0\rangle,\\ & \langle b,3\rangle,\langle b,2\rangle,\langle b,1\rangle,\langle b,0\rangle,\\ & \langle x,4\rangle,\langle x,3\rangle,\langle x,2\rangle,\langle x,1\rangle,\langle x,0\rangle\}\end{aligned}$$

As another example for the same computation, the set $\mathtt{past}_s(\{\langle b,3\rangle\})$ does not include any node with variable a. □

III. The Past Theorem

We now formalize the intuition that the values of the variables at a set T of nodes in a computation s depend only on the values at the nodes in $\mathtt{past}_s(T)$. We shall consider a set T all of whose nodes $\langle y,m'\rangle$ are at the same time m'. For any given earlier time $m < m'$, we construct a computation s' that coincides with s up to time m, in which the only changes after time m are those that appear in $\mathtt{past}_s(T)$, and the nodes of T obtain the same values in s' as they do in s.

Theorem 1: Fix an asynchronous circuit A, a computation s of A, times $m < m'$, and a set T of nodes $\langle y,m'\rangle$ at time m'. Then there is a computation s' of A such that $s'(t) = s(t)$ for all times $t \le m$, and for all variables x and times t in the range $m < t \le m'$, we have

(a) $s'_x(t) = s_x(t)$ if $\langle x,t\rangle \in \mathtt{past}_s(T)$ (and so, in particular, $s'_y(m') = s_y(m')$ for all $\langle y,m'\rangle \in T$), and

(b) $s'_x(t) = s_x(m)$ if $\langle x,m+1\rangle \notin \mathtt{past}_s(T)$.

Proof: We define the desired computation s' based on s. For times $t \leq m$ we define $s'(t) = s(t)$ as desired, while for times t such that $m \leq t < m'$, a PR with output variable z will fire at $s'(t)$ iff (i) $\langle z, t+1\rangle \in \mathtt{past}_s(T)$, (ii) the PR fires at $s(t)$ and (iii) the PR is enabled at $s'(t)$. Finally, from time m' on, the computation in s' proceeds in such a way that at every time step all enabled production rules fire.[2] Observe that s' is a valid computation of A, because by construction, only enabled PRs fire in s'. In addition, notice that by Locality, if $\langle x, m+1\rangle \notin \mathtt{past}_s(T)$ then $\langle x, t\rangle \notin \mathtt{past}_s(T)$ for all t in the range $m < t \leq m'$, and so, by definition of s', the gate for x does not fire between times m and $t-1$ in s'. It follows that $s'_x(t) = s_x(m)$ if $\langle x, m+1\rangle \notin \mathtt{past}_s(T)$, so that s' satisfies property (b). We now prove that s' satisfies property (a) for all times t in the range $m \leq t \leq m'$, by induction on t.

Base case $t = m$: By construction of s' we have that $s'_x(m) = s_x(m)$ for all variables x, including ones that satisfy $\langle x, t\rangle \in \mathtt{past}_s(T)$, so property (a) holds.

Inductive step $t > m$: Assume inductively that the claim holds for all variables z at time $t-1$, and let $\langle x, t\rangle \in \mathtt{past}_s(T)$. By locality, $\langle x, t-1\rangle \preceq_s \langle x, t\rangle$ and from $\langle x, t\rangle \in \mathtt{past}_s(T)$ we have that $\langle x, t-1\rangle \in \mathtt{past}_s(T)$ by transitivity of $\preceq_s$. Hence, by the inductive hypothesis $s'_x(t-1) = s_x(t-1)$. If x does not change at time $t-1$ in s then, by definition of s', the same is true at s', and so $s'_x(t) = s'_x(t-1) = s_x(t-1) = s_x(t)$. Assume that the value of x changes at time $t-1$ in s, and let B be the guard that causes the effective firing of x there. In particular, $s(t-1) \models B$. It suffices to show that $s'_z(t-1) = s_z(t-1)$ for every variable z that appears in B, since then $s'(t-1) \models B$, and by definition of s' there is an effective firing changing the value of x in s' as well. Let z be such a variable. Since there is an effective firing for x at $t-1$ in s, we have by the Successor clause that $\langle z, t-1\rangle \preceq_s \langle x, t\rangle$. Since $\langle x, t\rangle \in \mathtt{past}_s(T)$ we obtain by transitivity of $\preceq_s$ that $\langle z, t-1\rangle \in \mathtt{past}_s(T)$. Hence, $s'_z(t-1) = s_z(t-1)$ holds by the inductive hypothesis for z and $t-1$, and we are done. ■

Theorem 1, which we call the ***Past Theorem***, will be an essential tool in establishing formal properties of QDI circuits. It is based solely on the asynchrony of the circuits, and is true for other families of circuits, as well as for asynchronous distributed systems. Suppose that a firing of y at time t is allowed only if another variable, say z, has fired. Roughly speaking, Theorem 1 allows us to conclude that there must be a firing chain from z's firing to one on the inputs of y before time t. This captures an essential aspect of our formal reasoning about asynchronous circuits.

IV. Analysis of Isochronic Branches

Speed-independent (SI), QDI, and DI circuits operate correctly regardless of the delays on individual gates. In our definition of computations, an effective firing of a variable can be postponed by an arbitrary amount of time. This definition captures the arbitrary gate delay model used by SI, QDI, and DI circuits. In addition, QDI and DI circuits permit arbitrary delays on some (for QDI) or all (for DI) wire segments. The introduction of an arbitrary delay on a wire segment can be modeled by adding a buffer on the wire; i.e., by considering the two ends of the wire as separate variables (say x at the near end of the wire and $x^\dagger$ at the far end), and introducing a pair of production rules $x \mapsto x^\dagger\uparrow$ and $\neg x \mapsto x^\dagger\downarrow$ that account for the delay. This amounts to modifying the original circuit to analyze the impact of delays on wires. After this change, we need a way to relate computations in the original circuit to those in the modified circuit.

Let w be a computation of a circuit A' and suppose that V is a subset of the variables in A'. The ***restriction of*** w ***to*** the variables in V, denoted by $w|_V$, is defined to be the sequence of configurations over V given by $(w|_V)_x(t) = w_x(t)$ for every $x \in V$ and $t \geq 0$. Notice that if V is a strict subset of the variables in w, then moving from w to $w|_V$ can create new skip steps, since firings on some variables of w are ignored in $w|_V$. If circuit A' is obtained from A by adding variables, then we compare their computations by considering how stuttering-free variants of computations of A relate to stuttering-free variants of restricted computations of A':

Definition 3: Suppose that circuit A has variables V and circuit A' is obtained by adding variables to V and modifying the production rules in A. Let s be a computation of A and w a computation of A'. Then s is said to be ***consistent*** with w, denoted by $s \approx w$, if $\underline{s} = \underline{w|_V}$.

If the circuit A' is obtained from A by adding variables, s is a computation of A and w a computation of A', then $s \approx w$ implies that both computations perform the same changes to all variables of A, and do so in the exact same order. Indeed, if it is the case both that every computation of A is consistent with one of A' and that for every computation s' of A' there is a computation s of A consistent with s' then, in a precise sense, circuit A' exhibits all and only behaviors of the circuit A. In that case, we can view moving from A to A' as not modifying the behavior of the circuit A in an essential way.

A. Stability and Non-interference

We now describe the notions of ***stability*** and ***non-interference*** for asynchronous circuits. These notions are the same as those from [10] and are simply re-stated here using the terminology we have introduced.

Definition 4 (non-interference): A pair $B_u \mapsto z\uparrow$ and $B_d \mapsto z\downarrow$ of production rules in a circuit A is ***non-interfering*** if, for every computation s of A, there is no state $s(t)$ of the computation at which both $s(t) \models B_u$ and $s(t) \models B_d$. Both guards are never simultaneously enabled.

Definition 5 (stability): A PR $B \mapsto z\uparrow$ (respectively, $B \mapsto z\downarrow$) in a circuit A is said to be ***stable*** if for all computations s of A and for all times $t \geq 0$, if $s(t) \models B$ and $s(t+1) \models \neg B$, then $s_z(t+1) = 1$ (respectively, $s_z(t+1) = 0$).

In a QDI circuit, all production rules are stable and all gates consist of non-interfering PRs. We will use these two properties to reason about the impact of wire segment delays.

B. Isochronicity of Branches

In a QDI circuit, a wire segment (or branch) is non-isochronic if introducing an arbitrary delay on the wire segment does not modify the behaviors of the circuit. We now

[2] In fact, the computation could equally continue nondeterministically and fairly, if required.

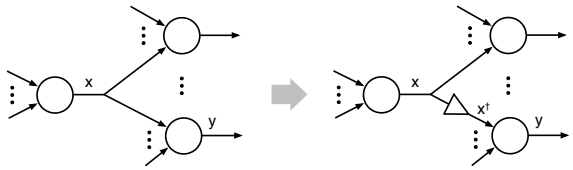

Fig. 3. Modification of original circuit A (on the left) to $A^\dagger$ (on the right) by the introduction of a single buffer on a branch from x to y.

examine the impact of introducing just one buffer, to model one wire segment.

Let x be a single variable in an asynchronous circuit A which is used in a guard for a production rule for variable y, and let $x^\dagger$ be a fresh variable that is not a variable of A. Consider a new asynchronous circuit $A^\dagger$ that is identical to A except for the following modification:

1) $A^\dagger$ contains two new production rules $x \mapsto x^\dagger\uparrow$ and $\neg x \mapsto x^\dagger\downarrow$;
2) The production rules for y use $x^\dagger$ instead of x; and
3) Variables x and $x^\dagger$ have the same value initially, so that $s^\dagger_x(0) = s^\dagger_{x^\dagger}(0)$ in all computations $s^\dagger$ of $A^\dagger$.

The modification is illustrated in Fig. 3.

$A^\dagger$ has all the same variables as A, and the additional variable $x^\dagger$. Since V is the set of variables in the circuit A, the set of variables in $A^\dagger$ is given by $V \cup \{x^\dagger\}$. The question we ask is: under what circumstances can we view computations of $A^\dagger$ as implementing computations of the original circuit A? It is straightforward to establish that all original computations of A still could occur in $A^\dagger$.

Lemma 2: For every computation s of A, there exists a computation $w^\dagger$ of $A^\dagger$ where $s \approx w^\dagger$.

Proof: To construct $w^\dagger$, whenever there is a non-x firing in the computation s of A we replicate it in the computation $w^\dagger$ of $A^\dagger$. If x changes in s, we replicate the firing in $w^\dagger$, but then follow that with a step consisting only of a firing of $x^\dagger$. ■

If the wire segment from x to y is a non-isochronic branch, then introducing the single buffer on the branch from x to y preserves stability of all gates in $A^\dagger$. We now show the consequence of requiring that the newly introduced buffer is stable.

*Theorem 2 (**firing loop**):* If the wire segment from x to y is a non-isochronic branch, then for every computation $w^\dagger$ of $A^\dagger$ where x changes at times t and $t' > t$, there is a chain of firings from $\langle x, t+1\rangle$ to $\langle x, t'+1\rangle$ in $w^\dagger$ that includes a change in $x^\dagger$.

Proof: Recall that if variable x changes value at time t, then its value at $t+1$ is different from the value at t. Let $w^\dagger$ be a computation of $A^\dagger$ in which x changes at times t and $t' > t$. By Theorem 1, setting $m = t$, $m' = t'+1$ and $T = \{\langle x, t'+1\rangle\}$ we have that there is a computation $u^\dagger$ of $A^\dagger$ in which (a) x changes value at times t and t', (b) $\mathtt{past}_{u^\dagger}(\langle x, t'+1\rangle) = \mathtt{past}_{w^\dagger}(\langle x, t'+1\rangle)$, and (c) all changes of variables that occur between times t and t' are in $\mathtt{past}_{u^\dagger}(\langle x, t'+1\rangle) = \mathtt{past}_{w^\dagger}(\langle x, t'+1\rangle)$. Moreover, all changes in values in $u^\dagger$ between times t and t' are changes that occur in $w^\dagger$ as well, and so it suffices to establish the existence of the desired firing chain in $u^\dagger$.

By stability of the gate for $x^\dagger$, the fact that x changed both at times t and $t' > t$ implies that $x^\dagger$ must change at some intermediate time t'' such that $t < t'' < t'$. Thus, by construction of $u^\dagger$ we have that $\langle x^\dagger, t''+1\rangle \preceq_{u^\dagger} \langle x, t'+1\rangle$. Since $x \neq x^\dagger$ we have by Lemma 1 that there must be a firing chain c from $\langle x^\dagger, t''+1\rangle$ to $\langle x, t'+1\rangle$ in $u^\dagger$. Moreover, since $x^\dagger$ is the output of a buffer whose sole input is x, the fact that $x^\dagger$ changes at t'' means that $\langle x, t''\rangle \hookrightarrow_{u^\dagger} \langle x^\dagger, t''+1\rangle$. Adding this firing of $x^\dagger$ at time t'' to the chain c, we obtain a firing chain in $u^\dagger$ from $\langle x, t''\rangle$ (and since $t < t''$ also from $\langle x, t+1\rangle$) to $\langle x, t'+1\rangle$ that includes a change in $x^\dagger$. The claim follows. ■

The production rules for all variables $z \neq y$ in $A^\dagger$ remain the same as in A, and those for y are obtained from the PRs for y in A by replacing all instances of x by the variable $x^\dagger$. Given a configuration $c^\dagger$ of $A^\dagger$, we can examine the configuration $c = c^\dagger|_V$ of A obtained by hiding variable $x^\dagger$. Clearly, every PR B_z for a variable $z \notin \{y, x^\dagger\}$ is true at $c^\dagger$ in $A^\dagger$ iff it is true at c in A. Moreover, if $c^\dagger_x = c^\dagger_{x^\dagger}$ then every PR B_y for y is true at $c^\dagger$ in $A^\dagger$ iff it is true at c in A.

C. Adversarial Firing Chains

For every computation s of A, there is a consistent $w^\dagger \approx s$ of $A^\dagger$, by Lemma 2. Therefore, the key question for us to ask is: are there computations in the new circuit $A^\dagger$ that do not correspond to any computation of A? In Theorem 3, we establish the following: if $A^\dagger$ has a computation $w^\dagger$ that is not consistent with any computation of A, then it must be that:

- The variable y fires in $w^\dagger$ at some time m' in a way that cannot be replicated in A;
- The reason for this mis-firing is that there is a firing chain in $w^\dagger$ from $\langle x, t\rangle$ to $\langle y, t'\rangle$ (for some $t < t'$), while $x^\dagger$ does not change between times t and t'. In other words, there is a computation where this firing chain is faster than the buffer on the branch from x to y; we call this an ***adversarial firing chain***.

The proof uses the following two main steps:

(i) The first step establishes that the earliest point at which the computations diverge is through a firing of y that cannot be replicated; furthermore, if this firing occurs at time m' then x and $x^\dagger$ must have different values at time m'.

(ii) The second and main step establishes the result, by showing that if the claim were not true, then there are computations of A where y is unstable.

Before we formally state the theorem and prove it, we extend the notion of consistency among computations to relate finite prefixes of computations.

Definition 6: Suppose that circuit $A^\dagger$ is obtained from A by adding a buffer with output variable $x^\dagger$ on the wire from x to y as described above. For a computation s of A and a computation $w^\dagger$ of $A^\dagger$, we write $s \sim_m w^\dagger$, and say that the two computations are ***compatible for** m **rounds***, if $s(t) = w^\dagger|_V(t)$ holds for all $t = 0, \ldots, m$.

We can now verify, by definition, that:

Lemma 3: Suppose circuit $A^\dagger$ is obtained from A by adding a buffer with output variable $x^\dagger$ on the wire from x to y as described above. For all computations $w^\dagger$ of $A^\dagger$, there exists a computation s of A such that $s \approx w^\dagger$ iff there exists a computation s' of A such that $s' \sim_m w^\dagger$ for all m.

Proof: Assume that $s \approx w^\dagger$. By Definition 3 we have $\underline{s} = \underline{w^\dagger|_V}$. Let $\beta : \mathbb{N} \to \{0,1\}$ be an indicator function recording where $w^\dagger|_V$ has non-skip steps; that is, $\beta(t) = 0$ if $w^\dagger|_V(t+1) = w^\dagger|_V(t)$, and $\beta(t) = 1$ otherwise. Denote $\mathsf{B}(t) = \sum_{i=0}^{t-1} \beta(i)$, the number of non-skip steps in $w^\dagger|_V$ by time t. Observe that $w^\dagger|_V(t) = \underline{w^\dagger|_V}(\mathsf{B}(t))$. Because $\underline{s} = \underline{w^\dagger|_V}$, we have that $w^\dagger|_V(t) = \underline{s}(\mathsf{B}(t))$. Let s' be defined by $s'(t) = \underline{s}(\mathsf{B}(t))$. By construction, (i) $s' = w^\dagger|_V$; (ii) $\underline{s}$ and s' only differ in skip steps, so s' is a computation of A iff s is a computation of A. Finally, since $s' = w^\dagger|_V$, we have that $s' \sim_m w^\dagger$ for all m (by Definition 6). The result follows. ■

We can now show our main result:

*Theorem 3 (**adversarial firing chain**):* Let A be a QDI circuit and let $A^\dagger$ be the circuit obtained by adding a buffer with output variable $x^\dagger$ on the wire from x to y in A. Suppose that $w^\dagger$ is a computation of $A^\dagger$ that is not consistent with any computation of A (i.e., $s \approx w^\dagger$ holds for no computation s of A). Then there is a firing chain in $w^\dagger$ from $\langle x, t\rangle$ to $\langle y, t'\rangle$ for some times $t < t'$ that does not include a firing of $x^\dagger$; in particular, $x^\dagger$ is unchanged between t and t' in $w^\dagger$.

Proof: Assume that there is no computation of A consistent with $w^\dagger$. By Lemma 3, that is equivalent to stating that there is no computation of A that is compatible with $w^\dagger$ for all rounds. Let $m' > 0$ be the largest time for which there exists some computation of A that is compatible with $w^\dagger$ for m' rounds. Moreover, let s be such a computation of A, for which $s \sim_{m'} w^\dagger$.

(i) Since m' is the largest such time, there must be an effective firing of a variable of A that takes place at time m' in $w^\dagger$, and does not take place at time m' of s. For every variable $z \neq y$ of A, any guard for z in A or $A^\dagger$ only depends on variables in V—which have the same values at $w^\dagger(m')$ and $s(m')$ because $s \sim_{m'} w^\dagger$. Hence, the only variable of A that can have an enabled effective firing in $w^\dagger(m')$ but not in $s(m')$ is y. Moreover, this is possible only if $s_x(m') \neq w^\dagger_{x^\dagger}(m')$. Since $s(m') = w^\dagger|_V(m')$, we also know that $w^\dagger_x(m') \neq w^\dagger_{x^\dagger}(m')$. Since $w^\dagger_x(0) = w^\dagger_{x^\dagger}(0)$ by assumption, there must be at least one firing of x in $w^\dagger$ before time m'.

Let $m < m'$ be the latest time before m' at which there is an effective firing of x in $w^\dagger$. Observe that this choice of m implies that $w^\dagger$ does not have an effective firing of $x^\dagger$ between times $m+1$ and m'. This is because (a) an effective firing of $x^\dagger$ would make $x = x^\dagger$; (b) m is the time of the last effective firing of x before m'; and (c) $w^\dagger_x(m') \neq w^\dagger_{x^\dagger}(m')$.

Two scenarios are possible: (SC-a) $w^\dagger_x(m) = w^\dagger_{x^\dagger}(m)$, variable x fires at time m, and there are no further changes to x or $x^\dagger$ until m'; or (SC-b) $w^\dagger_x(m) \neq w^\dagger_{x^\dagger}(m)$, *both* x and $x^\dagger$ fire at m, and they both remain unchanged until time m'.

(ii) We now show that $\langle x, m+1\rangle \preceq_{w^\dagger} \langle y, m'+1\rangle$. Assume by way of contradiction that this is not the case. We will show that this implies the existence of an unstable computation of A, violating the QDI property of A.

Let $B^\dagger_y$ denote the guard of the production rule of y in $A^\dagger$ that causes the effective firing at time m' in $w^\dagger$. Consider the set $T = \{\langle h, m'\rangle : h \text{ is an input to } B^\dagger_y \text{ in } A^\dagger\}$, and let $u^\dagger$ be the computation of $A^\dagger$ guaranteed to exist by Theorem 1, with respect to the computation $w^\dagger$, to this set T, and to the times m and m'. In particular,

- $u^\dagger(t) = w^\dagger(t)$ holds for $t = 0, \ldots, m$, which also means $s \sim_m u^\dagger$;
- the only changes in $u^\dagger$ after time m and up to m' are those that appear in $\mathtt{past}_{w^\dagger}(T)$; and
- $u^\dagger_h(m') = w^\dagger_h(m')$ holds for every $\langle h, m'\rangle \in T$.

Since by assumption $\langle x, m+1\rangle \not\preceq_{w^\dagger} \langle y, m'+1\rangle$ it follows that $\langle x, m+1\rangle \notin \mathtt{past}_{w^\dagger}(T)$. Therefore, by Theorem 1, there are no effective firings of x between times m and m' in $u^\dagger$. Since $x^\dagger$ is unchanged between $m+1$ and m' in $w^\dagger$, the construction of $u^\dagger$ by Theorem 1 ensures that $x^\dagger$ remains unchanged between $m+1$ and m' in $u^\dagger$ as well. Finally, by choice of m, there is an enabled effective firing of x at $u^\dagger(m)$. However, this firing is postponed by construction of $u^\dagger$ using Theorem 1.

Another useful observation is the following. Because $\langle x^\dagger, m'\rangle \in T$, we have by the definition of potential causality that $\langle x^\dagger, t\rangle \in \mathtt{past}_{w^\dagger}(T)$ for all $t \leq m'$. Thus, every effective firing of $x^\dagger$ at times $m+1$ to $m'-1$ in $w^\dagger$ occurs in $u^\dagger$ as well. Therefore, in scenario (SC-a) above, $u^\dagger_{x^\dagger}(t) = u^\dagger_x(t)$ for $t = m+1, \ldots, m'$ since the firing of x is postponed in $u^\dagger$. Also, in scenario (SC-b) above, $u^\dagger_{x^\dagger}(t) = u^\dagger_x(t)$ for $t = m+1, \ldots, m'$ because while the firing on x is postponed, the $x^\dagger$ firing at time m does occur in $u^\dagger$, making the two variables the same in state $u^\dagger(m+1)$. Hence $u^\dagger_x(t) = u^\dagger_{x^\dagger}(t)$ for $m < t \leq m'$.

We now construct a computation u of A that is compatible with $u^\dagger$ for m' rounds as follows. Let u be a computation of A in which $u(t) = s(t)$ for $t = 0, \ldots, m$. We construct the rest of u by induction on t for $t = m, \ldots, m'$. The induction hypothesis is that at time t, (a) precisely the variables of A that fire at time t in $u^\dagger$ can fire at time t in u; (b) there is an enabled effective firing of x at $u(t)$.

Base case $t = m$: Part (b) is true by construction at $t = m$. The guards for any variable $z \neq y$ in $A^\dagger$ only use variables in V, and hence any of those firings that occurs at m in $u^\dagger$ can be replicated in u at m because $u(m) = s(m) = u^\dagger|_V(m)$. If there is an effective firing of y in $u^\dagger$ at m, then we know that this firing also occurs in $w^\dagger$ at m. This means that the firing is enabled at $s(m)$ in A, because s and $w^\dagger$ are compatible until $m' > m$. Hence, the firing of y is also enabled at $u(m)$.

Induction step: Let $m < t < m'$ and assume inductively that the claim holds for t; we shall show the claim for $t+1$. Part (b) holds for $t+1$, because the original circuit A is stable and hence the enabled effective firing on x at $u(t)$ stays enabled at $u(t+1)$. All firings for $z \neq y$ can also be replicated as above. Finally, any firing of y can also be replicated, because

$u^\dagger_x(t) = u^\dagger_{x^\dagger}(t)$. Therefore, we have constructed a u such that $u \sim_{m'} u^\dagger$, and there is an enabled effective firing of x at $u(m')$.

Recall that in both u and $u^\dagger$ the variable x does not change between times m and m'. Thus, $u_x(m') = u^\dagger_x(m')$; using the relation between x and $x^\dagger$ in $u^\dagger$, we conclude that $u_x(m') = u^\dagger_{x^\dagger}(m')$. Key properties of the four different computations used in the proof, s, $w^\dagger$, $u^\dagger$, u, are highlighted in Table I.

Computation	**times** $\leq m$	**time** m	**time** $m+1$ **to** m'
s	all are compatible	x fires	compatible with $w^\dagger$
$w^\dagger$ (of $A^\dagger$)		x fires	compatible with s
$u^\dagger$ by Thm. 1		firing x postponed	x and $x^\dagger$ identical
u (of A)		firing x postponed	compatible with $u^\dagger$

TABLE I. THE FOUR COMPUTATIONS

Observe that the guard $B^\dagger_y$ is enabled at $w^\dagger(m')$. By construction of $u^\dagger$ in Theorem 1, the variables of T have the same values at $u^\dagger(m')$ as they do at $w^\dagger(m')$. Thus, the guard $B^\dagger_y$ is enabled at $u^\dagger(m')$ as well. Let B_y be the guard in A that corresponds to $B^\dagger_y$ in $A^\dagger$. Since $u^\dagger_x(m') = u^\dagger_{x^\dagger}(m')$, it follows that B_y is true at $u(m') = u^\dagger|_V(m')$.

By choice of s and m', B_y is false at $s(m')$ because y cannot have an effective firing in s at m'. We know that $s(m') = w^\dagger|_V(m')$, and $w^\dagger(m')$ agrees with $u^\dagger(m')$ on all variables in $B^\dagger_y$, and $u^\dagger|_V(m') = u(m')$. Hence, $u(m')$ and $s(m')$ agree on all variables in B_y other than x. Furthermore, since B_y is not enabled in $s(m')$, changing x in state $u(m')$ makes B_y false.

Finally, by the induction hypothesis used in the construction of u, there is an enabled effective firing of x at $u(m')$.

We have constructed a computation u of A where (a) the guard B_y is true at $u(m')$, enabling an effective firing of y there, (b) there is an enabled effective firing of x at $u(m')$, and (c) changing x at $u(m')$ disables B_y. Thus, y is unstable in A. However, by assumption, A is a QDI circuit, and so y must be stable—a contradiction to the assumption that $\langle x, m+1\rangle \not\preceq_{w^\dagger} \langle y, m'+1\rangle$. Setting $t = m+1$ and $t' = m'+1$, it follows that $\langle x, t\rangle \preceq_{w^\dagger} \langle y, t'\rangle$, and $x^\dagger$ does not change between t and t' in $w^\dagger$, as claimed. ■

Notice that this proof is based on two notions. That of stability, which underlies the definition of QDI circuits, and potential causality, captured by way of Theorem 1.

Since Theorem 3 shows that new, i.e., inconsistent, computations of $A^\dagger$ can only occur in a particular way, any argument or property that can be used to rule out the scenario established by Theorem 3 can be used to show that A and $A^\dagger$ have consistent computations. In particular, in a setting in which there are lower bounds on the time to complete firing chains and upper bounds on the delay of individual buffers, we have:

Corollary 1: Let $x^\dagger$ be an input to y. If the fastest adversarial firing chain from a change in x to y is slower than the delay of the buffer $x^\dagger$, then for every computation w of $A^\dagger$, there exists a computation s of A such that $w \approx s$.

Corollary 1 shows that satisfying the adversarial path condition is sufficient for the correct operation of QDI circuits. It is clear that this condition is necessary, because we can easily construct example circuits that fail to operate correctly when the adversarial path assumption is violated. This is precisely the necessary and sufficient timing condition shown in [6] for the correct operation of QDI circuits.

D. Identifying Isochronic Branches

As we now show, our mathematical foundations can provide guidelines for the practice of asynchronous circuit design. We know from Theorem 3 that the failure of an isochronic branch timing requirement results from an adversarial firing chain that is faster than the delay on the branch. Hence, it is important to identify all the isochronic forks in a circuit during the design process, so that the implementation can ensure that this timing constraint is met. As an illustrative example, we use Theorems 1 and 2 to provide a simple formal proof of a well-known type of isochronic fork.

Lemma 4: Let A be a circuit where the production rules for y are $G_u \mapsto y\uparrow$ and $(x \wedge G_d) \mapsto y\downarrow$, where both G_u and G_d do not depend on x. If x changes at least three times in a computation of A, then the branch from x to y is isochronic.

Proof: Suppose by way of contradiction that the branch is non-isochronic, and construct $A^\dagger$ from A by adding $x^\dagger$ as in Section IV-B. Let $s^\dagger$ be a computation of $A^\dagger$ in which x changes three times. Thus, there are times $m \geq 0$ and $m' > m$ with consecutive firings $x\downarrow$ at time m and $x\uparrow$ at m'. By Theorem 2, between these two changes in x there must be a firing loop that includes a change in $x^\dagger$. Since the only variable with input $x^\dagger$ is y, there must be a change in y at some time t satisfying $m < t < m'$. But $x^\dagger$ being false at $s^\dagger(m')$ means that the firing of $y\uparrow$ (the only possible firing of y on the firing loop) does not use the variable $x^\dagger$. Since the firing of $x^\dagger$ is not in the past of the variables in G_u at m', we can use Theorem 1 to construct an alternate computation where $x^\dagger$ is postponed—contradicting the firing loop requirement. Hence, the branch is isochronic. ■

V. DISCUSSION

A. Asynchronous Circuits and Isochronic Forks

Satisfaction of a set of delay requirements is an essential part of the correct operation of general asynchronous circuits. The "foam rubber wrapper" property was introduced using trace structures to characterize delay-insensitive communication [11]. The fact that researchers who were designing purely delay insensitive circuits were in fact making a timing assumption on wire forks was first highlighted in [10]. That paper introduced the notion of isochronic forks, arguing that they could be used to build more sophisticated asynchronous circuits than those possible under the assumption of purely delay-insensitive operation. [8] showed that the introduction of the isochronic fork was sufficient to build a circuit that could implement any computable function (modulo finite memory).

Many researchers initially viewed the isochronic fork requirement as a tight constraint on the delay of one branch of a fork. In particular, the assumption was seen as requiring that the delay through the branch of an isochronic fork was less than the delay through another branch plus the delay of the gate following the other branch. This led to concerns that isochronic forks were a challenging delay constraint (e.g. see [13]). Subsequently it was recognized that this constraint

is too strong, and a much weaker constraint is required for the correct operation of QDI circuits [3], [12]. More recent work [6] purported to prove that the isochronic fork requirement can be weakened to an adversarial path timing requirement. The current paper provides such a proof, and does so by making use of the well-established notion of potential causality from distributed systems. We expect the connection between asynchronous circuits and distributed systems to yield additional insights into the study of both fields.

B. Connection to Distributed Systems

As far as timing and coordination are concerned, there is a close connection between asynchronous circuits and asynchronous distributed systems. The potential causality relation $\preceq_s$ we defined in Section II-B is a variant of Lamport's *happened before* relation from his seminal paper [7]. The two differ in the Successor step, which in Lamport's case relates the sending of a message over an asynchronous channel to its delivery. While $\preceq_s$ corresponds to the existence of chains of firings as captured in Lemma 1, Lamport's relation corresponds to the existence of message chains between events at different sites. Message chains are the essential tool for creating information flow and coordinating actions in asynchronous distributed systems, just as firing chains are for asynchronous circuits. Indeed, Chandy and Misra showed in [1] that the only way a site can know about a change that occurs at another site of an asynchronous system is by way of a message chain. In a precise sense, firing chains can be shown to play the analogous role in asynchronous circuits. While we do not introduce a formal definition of what a node $\langle x, t\rangle$ ***knows*** about the circuit (this can be done using the framework of [4], [2] by viewing the circuit as an asynchronous system), it is interesting to consider our analysis in these terms. The fact that a firing chain is necessary in order to inform a node about the change of value of a different variable is essentially captured by Theorem 2.

VI. Conclusions

We adapt the notion of potential causality from the distributed systems literature, and apply it to study DI and QDI circuits. We state and prove the *past* theorem, which uses potential causality to capture an essential aspect of asynchrony in circuits. In a simple, trace-based model, we make use of the past theorem in order to obtain concise proofs of the firing loop theorem, which is part of the folklore, and of the isochronic branch theorem. Only a much more complex, and incomplete, proof of the latter result was previously available.

Our approach provides a foundation for future work on self-timed circuits since our theory can also incorporate other timing requirements. We plan to extend this approach to other timing models of asynchronous circuits, including bounded delay and relative timing. By using a single framework, we believe it will be possible to reason about hybrid asynchronous circuits—those that combine different timing disciplines, and we view this as a promising avenue for future research.

References

[1] K. Mani Chandy and Jay Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[2] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, 2003.

[3] Karl Fant. *Logically Determined Design*. Wiley, 2005.

[4] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[5] Sean Keller. Personal communication, November 2014.

[6] Sean Keller, Michael Katelman, and Alain J. Martin. A necessary and sufficient timing assumption for speed-independent circuits. In *Proceedings of the 15th IEEE International Symposium on Asynchronous Circuits & Systems (ASYNC '09)*, pages 65–76, 2009.

[7] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[8] Rajit Manohar and Alain J. Martin. Quasi delay-insensitive circuits are Turing-complete. In *Proceedings of the 2nd IEEE International Symposium on Asynchronous Circuits & Systems (ASYNC '96)*, 1996.

[9] Alain J Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[10] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In W. J. Dally, editor, *Proceedings of the 6th MIT conference on VLSI*, pages 263–278. MIT Press, 1990.

[11] Huub M. J. L. Schols. A formalisation of the foam rubber wrapper principle. Master's thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1985.

[12] Nattha Sretasereekul and Takashi Nanya. Eliminating isochronic-fork constraints in quasi-delay-insensitive circuits. In *ASP-DAC*. ACM, 2001.

[13] Kees van Berkel. Beware the isochronic fork. *Integration, the VLSI Journal*, 13(2):103–128, June 1992.

Appendix

We now give a formal definition of the stuttering-free variant $\underline{s}$ of a computation s and show that if s is a computation of circuit A, then so is $\underline{s}$. We proceed as follows. Given a computation s, we inductively define a function $\alpha_s : \mathbb{N} \to \mathbb{N}$ where, intuitively, $\alpha_s(k)$ will be the k'th non-skip step in s. For the base case, we define $\alpha_s(0) = 0$. Moreover, if s is a finite computation that performs only skip steps after time $\alpha_s(k)$, then $\alpha_s(k+1) = \alpha_s(k)$ (so that $\alpha_s(m) = \alpha_s(k)$ for all $m > k$). Otherwise,

$$\alpha_s(k+1) = \min\{k' > \alpha_s(k) : s(k') \neq s(k'-1)\}.$$

Observe that $\underline{s}(t) = s(\alpha_s(t)) = s(\alpha_s(t+1) - 1)$ holds by definition of α_s. Based on α_s, we can define a stuttering-free variant of the computation s, which we denote by $\underline{s}$, as follows:

$$\underline{s}(m) = s(\alpha_s(m)), \quad \text{for all } m \geq 0.$$

It is now easy to show:

Lemma 5: If s is a computation of a circuit A, then $\underline{s}$ is also a computation of A.

Proof: Fix the circuit A and let V be the set of its variables. First note that $\underline{s}(t)$ is a configuration of A for all $t \geq 0$, since $\underline{s}(t) = s(\alpha_s(t))$ by definition, and $s(\alpha_s(t))$ is a configuration of A for all t It remains to show that every change of value for a variable in $\underline{s}$ is a legal (enabled) firing. So assume that x has an effective firing at time t in $\underline{s}$. Then $\underline{s}_x(t) \neq \underline{s}_x(t+1)$. Since $\underline{s}(t) = s(\alpha_s(t+1) - 1)$ and $\underline{s}(t+1) = s(\alpha_s(t+1))$, the fact that x has an effective firing at time t in $\underline{s}$ implies that it has the same effective firing at time $\alpha_s(t+1) - 1$ in s. Since s is a computation of A, this firing must be enabled at $s(\alpha_s(t+1)-1)$. But $\underline{s}(t) = s(\alpha_s(t+1)-1)$, and hence the same firing is enabled at time t in $\underline{s}$ as well. We conclude that every effective firing in $\underline{s}$ is enabled when it occurs, and so $\underline{s}$ is a computation of A, as claimed. ■