Asynchronous Signalling Processes

Rajit Manohar*
Yale University
New Haven, CT 06520, USA
rajit.manohar@yale.edu

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

***Abstract*—A model of processes that interact via asynchronous wires carrying Boolean signals is presented. In this model, modules, called processes, can be made arbitrarily complex, can maintain local memory and can have an arbitrary number of inputs and outputs. A variety of circuit models can be represented by networks of signalling processes. It is shown that in a network of signalling processes consisting solely of single-ouput processes and forks, every module is an eventual C element. Consequently, the computational power of such a network is severely limited. This establishes that the celebrated C-element property of DI circuits follows solely from the fact that single output modules communicate over stable asynchronous wires. Conversely, it is shown that any Boolean function can be implemented using four-input/two-output processes where every process is either one gate (single output) or a pair of gates (two output).**

***Index Terms*—delay insensitive circuits**

I. Introduction

Researchers have studied a variety of models for delay-insensitive circuits as a way to decouple physical design concerns from logical design concerns. These models consisted of *modules* that were interconnected with delay-insensitive wires. As early as in the 1960s, the macromodular computer systems project aimed to develop modules so that "no logically irrelevant problems such as ... signal propagation delay" would have to be considered when assembling large-scale systems [1]. Various sets of components for delay-insensitive computation have been proposed, and they all include primitives that have more than one output wire. For example, Ebergen's model included the toggle primitive (with two outputs) as well as the $RCEL$ primitive (with three outputs) [2]. In an influential paper, Martin showed that every delay-insensitive circuit composed of single output gates that has only one computation consists entirely of C-elements [3]. More recent work extended the result to show that arbitrary delay-insensitive circuits composed of single output gates are also limited to eventually behave as C-elements [4].

There are two key assumptions underlying the C-element results. First, modules have a single output wire. Second, the modules are simple gates specified by production rules that model pull-up and pull-down logic networks. Both these assumptions are violated by the proposals in the literature for delay-insensitive modules. This naturally raises two key questions: (Q1) can we build useful delay-insensitive computations with complex modules, but that only have a single output? (Q2) if more than one output is needed, how many outputs are necessary and how complex do the modules have to be in order to support implementing arbitrary Boolean functions? In this paper we answer both these questions.

We propose *asynchronous signalling processes*, a general model for circuit modules interacting through delay-insensitive wires. In our model, modules can be made arbitrarily complex, and can have an arbitrary number of outputs. However, modules are connected by delay-insensitive wires that can be used to communicate binary values. We establish two main results. First, if modules are restricted to have single outputs, then the circuits remain extremely limited in the sense of [3], [4] despite the fact that modules can be arbitrarily sophisticated. In other words, the answer to question (Q1) is "no." Second, we show that two outputs are sufficient to make a fully general class of delay-insensitive circuits from the standpoint of computational power. In fact, we can use extremely simple modules to accomplish this so long as two outputs are permitted. In other words, the answer to (Q2) is that two outputs suffice, and the modules can be as simple as a pair of gates. This establishes that the key source of the limitations to delay-insensitivity shown in [3], [4] is the single output restriction.

II. Definitions and Model

A. Wires

The basic element used for communication between processes is a wire. A wire w consists of a pair of variables $(o_{\mathsf{w}}, i_{\mathsf{w}})$ in which o_{w} is called the wire's *originating variable* and i_{w} its *destination variable*. The variables o_{w} and i_{w} are considered *matching* variables. The former serves as an output of a process, and the latter is an input that the wire feeds into another process. The possible values of a destination variable are simply $\{0, 1\}$. The possible values of an originating variable are $\{0, 1, \mathsf{Z}\}$. When the value of the originating variable o_{w} is 0 and that of i_{w} is 1, we say that o_{w} is *enabled to signal* 0. Similarly, if o_{w} is 1 and i_{w} is 0, we say that o_{w} is *enabled to signal* 1. A signal is delivered over w in a transition that starts with an output variable o_{w} being enabled to signal, and ends with the value of its matching destination variable i_{w} assigned to the value that o_{w} had initially (while o_{w} remains

† Yoram Moses is the Israel Pollak academic chair at the Technion, and was supported in part by the BSF grant 2015820 which is coincident with NSF-BSF grant CCF 1617945.
* Rajit Manohar was supported in part by NSF-BSF grant CCF 1617945, DARPA IDEA grant FA8650-18-2-7850, and in part by DARPA POSH grant HR001117S0054-FP-042.

unchanged). When o_{w} has value Z it is disabled, and no change in the destination variable can take place.[1]

Given a set of originating variables O, we denote by S^O the set of possible assignments $\sigma_O \colon O \to \{\mathsf{0}, \mathsf{1}, \mathsf{Z}\}$. Similarly, given a set of destination variables I, we denote by S^I the set of possible assignments $\sigma_I \colon I \to \{0, 1\}$.

B. Processes

A *process* in our framework is associated with a set $\mathtt{Q}$ of internal states, a set O of originating variables called its outputs, and a set I of destination variables called its inputs. A *local state* of such a process consists of a triple $\ell = (\mathsf{q}, \sigma_O, \sigma_I)$, where $\mathsf{q} \in \mathtt{Q}$ is the internal state, $\sigma_O \colon O \to \{0, 1, Z\}$ maps output variables to their value, and $\sigma_I \colon I \to \{0, 1\}$ maps input variables to their values.

Formally, a *process* in our framework is a tuple $p = (\mathtt{Q}, \hat{\delta}, O, I)$ that describes a nondeterministic automaton where $\mathtt{Q}$, O, and I are as above, and $\hat{\delta} \colon \mathtt{Q} \times S^O \times S^I \to (2^{\mathtt{Q} \times S^O} \setminus \emptyset)$ is its nondeterministic transition function. Intuitively, $\hat{\delta}$ sets the next internal state and the next assignment to the output variables of a process p, based on p's current local state.[2] A *step* of the process changes its local state from $(\mathsf{q}, \sigma_O, \sigma_I)$ to $(\mathsf{q}', \sigma'_O, \sigma_I)$ where $(\mathsf{q}', \sigma'_O) \in \hat{\delta}(\mathsf{q}, \sigma_O, \sigma_I)$. In general, different outputs of a process can have distinct values; i.e., each output of a process can depend on its inputs and its internal state in a different way.

In circuits, a single signal often has multiple destinations. To conveniently model this scenario, we define a special type of process called a *fork* that has a single input variable and multiple output variables. At every step, a fork simply copies the value of its (single) input to all of its outputs.

Given a set of processes P, a *configuration* c of P assigns a local state $\ell = (\mathsf{q}, \sigma_O, \sigma_I)$ to every process $p \in P$.

C. Networks of Signalling Processes

An *SP network* is a triple $\mathcal{S} = (P, W, C_0)$ consisting of a set of processes P, a set W of the wires appearing in P, and a set C_0 of initial configurations of P. To ensure that W represents the input/output variables in P, we require:

1) Every input and output variable in a process $p \in P$ belongs to a wire in W, and
2) For every wire $\mathsf{w} = (o_{\mathsf{w}}, i_{\mathsf{w}}) \in W$ there are two unique processes $p, p' \in P$ such that $o_{\mathsf{w}} \in O^p$ and $i_{\mathsf{w}} \in I^{p'}$.

It is sometimes convenient to consider the communication graph $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ underlying an SP network $\mathcal{S} = (P, W, C_0)$. This is a directed graph with vertex set $V_{\mathcal{S}} = P$ consisting of the processes of $\mathcal{S}$, and where there is a directed edge $(q, p) \in E_{\mathcal{S}}$ if there are matching output and input variables $o_{\mathsf{w}} \in O^q$ and $i_{\mathsf{w}} \in I^p$. The associated wire w is considered an *output wire* of q and an *input wire* of p. Note that while a process typically has external inputs and outputs, an SP network has neither. In this sense, a process may be open to external interaction, but an SP system is closed.

D. Computations

A *computation* of an SP network $\mathcal{S}$ is a sequence $s = c_0, c_1, c_2, \ldots$ of configurations of $\mathcal{S}$, in which (i) c_0 is an initial configuration of $\mathcal{S}$; and (ii) For $k \geq 0$, configuration c_{k+1} is obtained from c_k by having every process p in $\mathcal{S}$ take a step, and then having a (possibly empty) set of signals that are enabled in c_k delivered over their wires.

For a computation $s = c_0, c_1, c_2, \ldots$, we use $s(t)$ to denote the configuration c_t, and we use $s_p(t)$ to denote the local state $(\mathsf{q}, \sigma_O, \sigma_I)$ of process p in configuration c_t. Let w be a wire with output variable o_{w} and (matching) input variable i_{w}. A signal is delivered over the wire w at a configuration $s(t)$ of a computation s if $t > 0$, the output variable o_{w} is enabled at time $t-1$ (i.e., $s(t-1)(i_{\mathsf{w}}) \neq s(t-1)(o_{\mathsf{w}}) \neq \mathsf{Z}$), and $s(t)(i_{\mathsf{w}}) = s(t)(o_{\mathsf{w}}) = s(t-1)(o_{\mathsf{w}})$. (Thus, a signal that is delivered at time t is already reflected in the input variable of the receiving process at time t; the step taken by the process at time t can, in general depend on the signal's updated value.)

E. Stability

In analogy to circuits, signalling networks must be designed in such a way that their wires do not suffer from glitches and short circuits. It is thus the responsibility of the designer of a signalling network $\mathcal{S}$ to ensure that every computation s of $\mathcal{S}$ satisfies the following condition:

- *Stability*. For every pair o_{w} and i_{w} of matching variables in $\mathcal{S}$ and all times $t \geq 0$, if $s(t)(o_{\mathsf{w}}) \neq \mathsf{Z}$ and $s(t+1)(o_{\mathsf{w}}) \neq s(t)(o_{\mathsf{w}})$, then $s(t+1)(i_{\mathsf{w}}) = s(t)(o_{\mathsf{w}})$.

Stability ensures that once an output variable is set to $\mathsf{0}$ (or $\mathsf{1}$), this value does not change before it propagates to the matching input variable.

When using signalling networks to describe asynchronous circuits, it is natural to assume that all wires are stable. For the remainder of the paper, all signalling process networks that we consider will be assumed to satisfy stability.

F. Asynchronous Circuits as SP Networks

There are many possible computations of a given SP network that begin with an initial configuration c_0. The assumption that an arbitrary set of enabled signals can be delivered at any step of a computation of an SP network makes interprocess wires asynchronous.

SP systems can be used to model popular asynchronous circuit models. The class of purely delay-insensitive (DI) circuits can be directly captured by mapping gates to processes with a single state each, where the function $\hat{\delta}$ for a gate captures the behavior of the gate's pull-up and pull-down switching logic.

All other asynchronous circuit families introduce timing assumptions either on wires, on gates, or both. In an SP network, it is convenient to use the transition function $\hat{\delta}$ within

[1] We have chosen not to have the originating variable value Z carry the information of whether the last value "delivered" on the wire was $\mathsf{0}$ or $\mathsf{1}$. If necessary, this information can be captured by the internal state in our formalism.

[2] Since we allow the transition function to be nondeterministic, our processes can correspond to a rich variety of devices, including synchronous and asynchronous gates or sub-circuits.

an individual process to capture any delay assumptions. For example, quasi delay-insensitive (QDI) circuits have isochronic fork constraints. If the output of a gate has an isochronic fork to a set of target gates, those gates could be grouped into a single process in an SP network so that the fork is encapsulated within a individual SP process. Note that since QDI circuits are typically decomposed into small modules that communicate over delay-insensitive wires, this procedure of merging gates would typically end at module boundaries. In the resulting SP network, every process would correspond to a module.

More generally, large-scale self-timed circuits are typically designed as a collection of cooperating modules that exchange information via communication channels. Often channels between modules use delay-insensitive protocols. In this scenario, each circuit module corresponds to an SP process, and wires connecting modules correponds to wires in the SP network. Each process' transition function $\hat{\delta}$ can capture the functional behavior of the corresponding module for any set of underlying timing assumptions in the circuit implementation. Hence, SP networks can capture the behavior of any asynchronous circuit family where modules are connected by delay-insensitive wires.

Bundled-data communication includes a relative timing constraint between the control and data wires in the communication link. The asynchronous SP model does not capture this in a useful way, since all the wires and control signals would have to be encapsulated within a single process. We discuss how a small circuit transformation can be used to extend the asynchronous SP model to capture the behavior of such circuits in Section VI.

In summary, the SP model is applicable whenever an asynchronous circuit implemented using any model can be partitioned into components that communicate over delay-insensitive wires.

III. Asynchronous Signalling Processes

A. Potential Causality

The sequence defined by a computation is indexed by an integer, which plays the role of an external notion of time. (We use both t's and m's for these indexes.) But, in the asynchronous setting that we are considering in this section, the signalling processes in the system do not have access to the current time, and it does not affect their operation. Reasoning from the outside, we will be interested in distinguishing the information available to a process at different times t, and in particular how this information propagates through the input and output variables. For this purpose, given a process p, a pair $\langle p, t\rangle$ is called a ***node***. We use such a node to refer to process p at time t.

Let p and q be two processes connected by a wire w where o_{w} is an output variable of p and i_{w} is its matching input variable in q. If a signal from p is delivered to q over w at time t in a given computation s, then we write $\langle p, t\rangle \hookrightarrow_s \langle q, t+1\rangle$ and call $\langle q, t+1\rangle$ a *successor* of $\langle p, t\rangle$ in s. Following [5], [6], given a computation s we define a partial order $\prec_s$ over nodes called ***potential causality*** to be the minimal relation satisfying the following three conditions:[3]

Locality: $\langle p, t\rangle \prec_s \langle p, t'\rangle$ if $t < t'$;

Successor: $\langle p, t\rangle \prec_s \langle q, t+1\rangle$ if $\langle p, t\rangle \hookrightarrow_s \langle q, t+1\rangle$;

Transitivity: $\langle p, t\rangle \prec_s \langle q, t'\rangle$ if both $\langle p, t\rangle \prec_s \langle r, m\rangle$ and $\langle r, m+1\rangle \prec_s \langle q, t'\rangle$, for some node $\langle q, m\rangle$.

When $\langle p, t\rangle \prec_s \langle q, t'\rangle$ where $p \neq q$, the definition of potential causality implies that there must be a finite sequence of sent signals that relate $\langle p, t\rangle$ to $\langle q, t'\rangle$. This is captured by the notion of *signalling chains*.

Definition 1: We say that there is a ***signalling chain from*** $\langle p, t\rangle$ ***to*** $\langle q, t'\rangle$ ***in the computation*** $\mathbf{s}$ if there is a sequence of processes $\langle x_1, \ldots, x_k\rangle$ with $x_k = q$, and a sequence of times $\langle t_1, \ldots, t_k\rangle$ with $t \leq t_1$, $t_k < t'$, and $t_{i-1} + 2 \leq t_i$ for $2 \leq i \leq k$, such that $\langle p, t_1\rangle \hookrightarrow_s \langle x_1, t_1 + 1\rangle$ and $\langle x_{i-1}, t_i\rangle \hookrightarrow_s \langle x_i, t_i + 1\rangle$ for $2 \leq i \leq k$.

The notion of a signalling chain is analogous to a message chain in a distributed computation, and to a firing chain in asynchronous circuits [3]. The fact that '$\prec_s$' is the minimal relation satisfying the Locality, Successor and Transitivity conditions means that $\langle p, t\rangle \prec_s \langle q, t'\rangle$ holds only if it can be derived by a finite number of applications of these conditions. Based on this, a rather straightforward consequence of the definition of $\prec_s$ is the following lemma, whose proof is delegated to the Appendix:

Lemma 1: For processes $p \neq q$, if $\langle p, t\rangle \prec_s \langle q, t'\rangle$ then both $t < t'$ and there is a signalling chain from $\langle p, t\rangle$ to $\langle q, t'\rangle$ in s.

Lemma 1 immediately implies that if $\langle p, t\rangle \prec_s \langle q, t'\rangle$ then there is a path from p to q in the SP network graph.

Recall that a node $\langle p, t\rangle$ refers to process p at time t. Intuitively, the '$\prec_s$' relation captures potential causality within the computation s in the sense that information at a node $\langle p, t\rangle$ can affect the state of q at time t' in s only if $\langle p, t\rangle \prec_s \langle q, t'\rangle$. I.e., if $\langle p, t\rangle \not\prec_s \langle q, t'\rangle$ then the state of p at time t *can not* influence the information and actions of q at time t' in s. This is made precise by the following result.

*Theorem 1 (**Past Theorem**):* Fix a system $\mathcal{S}$ of signalling processes, times $m < m'$, a computation s of $\mathcal{S}$ and a node $\langle q, m'\rangle$. There is a computation s' of $\mathcal{S}$ such that $s'(t) = s(t)$ for all times $t \leq m$, and for all processes p and times t in the range $m < t \leq m'$, we have that

(a) $s'_q(m') = s_q(m')$,
(b) if $\langle p, t\rangle \prec_s \langle q, m'\rangle$ then $s'_p(t) = s_p(t)$, and
(c) if $\langle p, t\rangle \not\prec_s \langle q, m'\rangle$ then p receives no signal at time t.

[3] A fourth condition could be added, to account for the fact that a process step transfers information from the input variables to the output variables of the process. Instead, the transitivity condition allows for one step between time m and time $m+1$ at the intermediate process r, in which this transfer can occur. We note that the transitivity condition in the relation $\preceq_s$ relation defined in [6] would not faithfully apply to signalling processes. We choose to have $\prec_s$ be a strict (non-reflexive) relation, whereas the corresponding condition $\preceq_s$ of [6] is reflexive, simply for ease of exposition.

Proof: We define the desired computation s' based on s. For times $t \leq m$ we simply define $s'(t) = s(t)$ as desired. For times t in the range $m < t \leq m'$,

(i) if $\langle p, t\rangle \prec_s \langle q, m'\rangle$ or $\langle p, t\rangle = \langle q, m'\rangle$ then $s'_p(t) = s_p(t)$; while

(ii) if $\langle p, t\rangle \not\prec_s \langle q, m'\rangle$ then at time t in the computation s' process p receives no signal on any of its incoming wires, and performs an arbitrary legal step (according to its local state $s'(t)$ and its transition function $\hat{\delta}$).

Finally, from time $m' + 1$ on, the computation in s' proceeds in such a way that at every time t, each of the processes takes an arbitrary legal step, and every enabled signal is delivered.

By construction of s' we clearly have that $s'(t) = s(t)$ for all times $t \leq m$. Moreover, conditions (i) and (ii) immediately imply that claims (a), (b) and (c) hold. To complete the proof, we need to show that s' is a legal computation of the SP system $\mathcal{S}$. Since $s'(0) = s(0)$ the new computation s' starts in a legal initial configuration of $\mathcal{S}$. The next property we need to show is that every configuration in s' is obtained from its predecessor by having every process p taking a legal step, and a possibly empty set of enabled signals delivered over their wires. By definition of s', this immediately holds for all configurations $s'(t)$ for $0 < t \leq m'$, and for all $t > m'$. To establish the claim for t in the range $m < t \leq m'$, it suffices to show that for every p such that $\langle p, t\rangle \prec_s \langle q, m'\rangle$ we have that every signal that is delivered to p at time t of s is enabled to be delivered to p at time t in s' as well. Recall $s'_p(t) = s_p(t)$ for such p and t, by definition of s'. If the delivered signals in s are enabled in s' as stated, then the same transition performed by $\hat{\delta}_p$ at $s_p(t-1)$ can be applied at $s'_p(t-1)$, and all signals delivered to p at time t in s can be delivered in s'. It follows that the transition from $s'_p(t-1)$ to $s'_p(t) = s_p(t)$ is legal.

We now prove for all t in the range $m < t \leq m'$, that if $\langle p, t+1\rangle \prec_s \langle q, m'\rangle$ or $\langle p, t\rangle = \langle q, m'\rangle$ then every signal that is delivered to p at time t of s is enabled at $s'(t-1)$ to be delivered to p.

Case $t = m + 1$: In this case, $t - 1 = m$. By construction of s' we have that $s'(m) = s(m)$ and so, the same signals are enabled at $s'(t-1)$ and at $s(t-1)$, ensuring that the condition holds.

Case $m' \geq t > m + 1$: Assume that a signal is delivered on w from p' to p at time t in s. Thus, $\langle p', t-1\rangle \hookrightarrow_s \langle p, t\rangle$, and so by Successor we have that $\langle p', t-1\rangle \prec_s \langle p, t\rangle$. In case $\langle p, t\rangle = \langle q, m'\rangle$ we clearly have that $\langle p', t-1\rangle \prec_s \langle q, m'\rangle$. Otherwise, since $\langle p, t+1\rangle \prec_s \langle q, m'\rangle$ it follows by Transitivity that $\langle p' - 1, t\rangle \prec_s \langle q, m'\rangle$. Thus, by definition of s', we have that $s'_{p'}(t-1) = s_{p'}(t-1)$. By Locality we have that $\langle p, t-1\rangle \prec_s \langle p, t\rangle$, and it similarly follows that $s'_p(t-1) = s_p(t-1)$. Thus, the variables o_w of p' and i_w of p have the same values at time $t-1$ in s' as they do in s, and since o_w is enabled in $s(t-1)$ it is enabled in $s'(t-1)$ as well, and the claim follows.

□

We note that the Past Theorem is a direct generalization of the corresponding Theorem 1 in [4], from DI circuits to signalling processes. The statement is slightly modified, but the proof in our case needed to be modified to account for the fact that processes need not be asynchronous. In both cases, the theorem does not depend on the stability property. Combining the stability property with the Past Theorem, we can obtain the following analogue of [4]'s Theorem 2:

*Theorem 2 (**feedback**):* Let s be a computation of an SP system $\mathcal{S}$, and let q be a process in $\mathcal{S}$ with output variable $o_\mathsf{w} \in O^q$ driving the wire w. Moreover, let $m' > m$. Finally, assume that o_w is enabled to signal a value $v \neq \mathsf{Z}$ at $s(m)$, and that $s(m')(o_\mathsf{w}) \neq s(m)(o_\mathsf{w})$. Then there is a signalling chain from $\langle q, m\rangle$ to $\langle q, m'\rangle$ in s whose first signal is sent over w.

Proof: Assume that s, q, o_w, m and m' satisfy the conditions of the theorem. Moreover, let $i_\mathsf{w} \in I^p$ be the input variable matching o_w in $\mathcal{S}$. Let s' be the computation of $\mathcal{S}$ that is guaranteed by Theorem 1 with respect to the computation s, times $m < m'$, and the node $\langle q, m'\rangle$. We start by showing that the value v is delivered over w to p at some time $t' \in [m+1, m']$. By Theorem 1 the computation s' satisfies that $s'(m) = s(m)$ and that $s'_p(t) = s_p(t)$ for all $t \leq m'$. Since o_w is enabled to signal v at $s(m)$, the same is clearly true at $s'(m)$. Moreover, since $s'(m') = s(m')$ we have that $s'(m')(o_\mathsf{w}) \neq s'(m)(o_\mathsf{w})$. Thus, $s'(m)(i_\mathsf{w}) \neq s'(m)(o_\mathsf{w})$. Moreover, for some $t \in [m, m'-1]$ we have that $s'(t)(o_\mathsf{w}) = s'(m)(o_\mathsf{w}) \neq \mathsf{Z}$ and $s'(t+1)(o_\mathsf{w}) \neq s'(t)(o_\mathsf{w})$. Without loss of generality, suppose that t is the earliest such time. Since o_w is assumed to be stable, we have by stability that $s'(t+1)(i_\mathsf{w}) = s'(t)(o_\mathsf{w}) = s'(m)(o_\mathsf{w}) = v$. Since i_w is a binary variable, $s'(m)(i_\mathsf{w}) = 1 - v$ and $s'(t+1)(i_\mathsf{w}) = v$, it follows that there is some $t' \in [m+1, m']$ at which the signal v is delivered to p in s' over w, as claimed.

By definition of s', a signal can be delivered to p over w at time $t' \in [m+1, m']$ in s' only if it is delivered to p over w at time t' in s, and $\langle p, t'\rangle \prec_s \langle q, m'\rangle$. Hence, $\langle q, t'-1\rangle \hookrightarrow_s \langle p, t'\rangle$, and by Lemma 1, there is a signalling chain from $\langle p, t'\rangle$ to $\langle q, m'\rangle$ in s. Since $t' > m$, it follows that there is a signalling chain from $\langle q, m\rangle$ to $\langle q, m'\rangle$ in s whose first signal is sent over w. □

Theorem 2 shows that, in any SP network, if an output variable in process q is enabled to signal a change at a certain time and the output variable changes at a later time, there must be a sequence of signals—a signalling chain—that begins with this signal from q and continues in a loop back to q. Intuitively, q's output can only change its state once it is informed (through an input to q) that its output change has been received. What is interesting is that the processes in the SP network might be implemented with asynchronous circuits that make arbitrary internal timing assumptions within processes. What matters is that the global communication between processes uses delay-insensitive wires that are stable; nothing else is required to ensure that the feedback loop exists.

IV. Single-output SP Systems

A ***single-output SP system*** is one in which every process is either a fork, or has a single output wire. This is a natural

generalization of DI circuits with single-output gates. Except, of course, that the processes in a DI circuit are gates that have no memory, whereas those in an SP system can be arbitrarily complex finite-state machines. Our purpose in this section is to show that despite the added computational power of processes in SP systems compared to gates in DI circuits, the single-output assumption makes them as limited as circuits composed of single-output gates are. In essence, the analysis in [4] can be performed in the more general setting, showing that every process in a single-output SP system behaves like an eventual C-element.

Definition 2 (Successor property): An SP system ***$\mathcal{S}$ exhibits the successor property*** iff for every computation s of $\mathcal{S}$ and process p, if p receives signals on an (input) wire w at times t and $t^\dagger > t$ in s, then a signal is sent on p's (single) output wire at some time $t' \in [t+1, t^\dagger - 1]$.

We can use Theorem 2 to show the following.

Lemma 2: Every single-output signalling system exhibits the successor property.

Proof: Let $\mathcal{S}$ be a single-output signalling system, and let p be a process in $\mathcal{S}$. Moreover, let s be a computation of $\mathcal{S}$ in which p receives signals on an (input) wire w at times t and $t^\dagger > t$. Denote by i_w process p's input variable on w, and by q the process whose output variable o_w matches i_w. Since p receives signals on w at t and at $t^\dagger$, we have that o_w is enabled to signal at $s(t-1)$ and at $s(t^\dagger - 1)$. It follows that $s(m')(o_\mathsf{w}) \neq s(t-1)(o_\mathsf{w})$, for some time $m' \leq t^\dagger - 1$. By Theorem 2 there is a signalling chain from $\langle q, t-1\rangle$ to $\langle q, t^\dagger - 1\rangle$ in s whose first signalling occurs over o_w. Since $p \neq q$, this chain contains at least two signals. The first signal in this chain reaches p at a time $m \geq t$, and the second signal is sent on p's (single) output wire at some time $t' \in [t+1, t^\dagger - 1]$. It follows that $\mathcal{S}$ satisfies the successor property, as claimed. □

Suppose that p is a single-output process, with input wire w whose output variable belongs to process q in an SP network $\mathcal{S}$. We call the wire w a ***feedback input wire*** of p if, in some computation s of $\mathcal{S}$, process p receives signals more than once over w. An immediate consequence of Theorem 1 is that if w is a feedback input wire of p, then there is a (simple) cycle containing p in the SP network graph $G_\mathcal{S}$, whose last edge is (q, p). Moreover, if p's output wire is w', then Lemma 2 implies that between every two signals that are delivered over w, at least one signal is delivered over w'.

Given Lemma 2, we can modify the argument in [4] and show that the single-output property implies that processes behave in a manner resembling C elements. The following definitions formalizes the condition that processes satisfy in such circumstances.

Definition 3 (Eventual C-element): A process p in an SP system $\mathcal{S}$ is called an ***eventual C-element*** if for every computation s of $\mathcal{S}$ there exists a time $T = T(s)$ such that the following holds for every output wire w of p: If signals are delivered over w both at a time $t \geq T$ and at a time $t^\dagger > t$ in s, then for every feedback input wire w' of p there is a time t' in the range $t < t' < t^\dagger$ at which p receives a signal on w' in s.

Theorem 3: In a single-output SP system, every process is an eventual C-element.

Proof: Fix a single-output SP network $\mathcal{S}$, a computation s of $\mathcal{S}$, and a non-fork process p with output wire w. Moreover, fix a feedback input wire w' of p, originating from process p'. To prove the claim, it suffices to show that there is a time $t_{s,\mathsf{w}'}$ after which signals are received by p over w' and are sent by p over w in alternating order. The time $t = T(s)$ required by the definition of an eventual C-element process would just be the maximum $t_{s,\mathsf{w}'}$ over all of p's feedback wires w'. By Lemma 2 we have that between any two signals that p receives over w' in s, at least one signal is delivered over its output wire w. It remains to show that the converse is true from some point on: Between every pair of signals delivered over the output wire w, one is received over the input wire w'. The remainder of the proof follows the argument in the proof of Lemma 6 from [4].

Since w' is a feedback input wire of p, there is a simple cycle $\langle p, p_1, \ldots, p_d, p', p\rangle$ in $G_\mathcal{S}$, for some finite $d \geq 0$. Denote $Q = \{p, p_1, \ldots, p_d, p'\}$. For every $q \in Q$, we denote by q^+ the successor of q in the cycle, and by q^- its predecessor. In the remainder of the proof, whenever we say that a process $q \in Q$ *receives a signal* this is shorthand for a signal being delivered on the wire from q^- to q. Fix a computation s of $\mathcal{S}$. The successor property captured by Lemma 2 implies that a necessary condition for $q \in Q$ to receive a signal at $s(m)$ is that q^+ received a signal since the last time $m' < m$ at which q received a signal in s (provided that such an earlier time m' exists).

For every time $m \geq 0$, define

$$W(m) = \big\{q \in Q : \text{ for every } m' < m, \text{ if } q \text{ received a signal at } s(m'), \text{ then } q^+ \text{ received a signal at least once between } m'+1 \text{ and } m-1 \text{ in } s.\ \big\}$$

Thus, $W(m)$ is the set of processes in the cycle that are allowed to receive a signal at time m, as far as the successor property is concerned. Clearly, $W(0) = Q$. If $q \in Q$ receives a signal at a given time m, then its predecessor in the cycle will be added to W, unless it is in $W(m)$ already. Let $\mathsf{received}(m)$ be the set of processes in Q that receive a signal at $s(m)$. The rule by which W is updated is:

$$W(m+1) = \Big(W(m) \cup \{q^- : q \in \mathsf{received}(m)\}\Big) \setminus \mathsf{received}(m). \tag{1}$$

The processes q that receive signals are removed from W, and their predecessors q^- are added to W. (Except that if the predecessor q^- also receives a signal at m, then q^- will not be added to W.) Equation (1) implies that W cannot grow in size: $|W(m+1)| \leq |W(m)|$ for all $m \geq 0$. Indeed, $|W(\cdot)|$ remains the same only if none of the predecessors q^- are in $W(m)$. Moreover, if $W(m)$ is a singleton, then $W(m')$ is a singleton for all $m' > m$, since q can receive a signal only if $W(m) = \{q\}$, but then $W(m+1) = \{q^-\}$, since $q^- \notin W(m)$.

So suppose that some $q \in Q$ receives signals at $s(m)$ and at $s(m')$ with $m < m'$, and its predecessor q^- does receive a signal in the interim. We will show that $|W(m'+1)| < |W(m)|$. If its predecessor q^- also receives a signal at $s(m)$, then both q and q^- are removed from $W(m)$ and at best $(q^-)^-$ is added on their account, so $|W(m+1)| \leq |W(m)| - 1 < |W(m)|$. By monotonicity of $|W(\cdot)|$ we have that $|W(m'+1)| \leq |W(m+1)| < |W(m)|$, as claimed, since $m' > m$. Otherwise q^- does not receive a signal at $s(m)$, and by Equation (1) we have that $q^- \in W(m+1)$ because q received a signal at $s(m)$. Moreover, $q^- \in W(m')$ since q^- does not receive a signal between $m+1$ and m'. Indeed, both $q^-, q \in W(m')$ since $q \in \mathsf{received}(m')$. Equation (1) now implies that $|W(m'+1)| < |W(m')|$, and by monotonicity of $|W(\cdot)|$ we have that $|W(m'+1)| < |W(m')| \leq |W(m)|$, establishing the claim.

It follows that the total number of times at which a process in Q receives two signals in s without its predecessor receiving a signal in the interim is no greater than $d = |Q|-2$, and is thus finite. Given Lemma 2, we obtain that there is a time $t_{s,y}$ after which signals over w' and over w are delivered in alternating order, and we are done. □

The main takeaway from this result is that the limitations on circuits imposed by delay-insensitivity are a consequence of the single-output constraint combined with the fact that wires are assumed to be asynchronous. The complexity of the gate does not play a role, since our processes can model arbitrarily complex finite state machines.

V. Simple Two-output Systems are Universal

A ***two-output SP system*** is one in which every process is either a fork, or has at most two outputs. In this section, we show that adding a second output is sufficient to implement any Boolean function. Furthermore, our construction uses SP processses without any internal memory (i.e., a single state), and with at most four inputs. This shows that extremely simple SP processes with two outputs are sufficient to implement arbitrary computation.

Relaxing the delay-insensitivity requirement on wires by introducing the isochronic fork assumption (as suggested in [3]) is one way to circumvent the limitations introduced by the single output assumption [7]. Intuitively, an isochronic fork is a fork where one branch of the fork is assumed to be at least as fast as the other. Isochronic forks can be used in a wide range of contexts, including scenarios where a branch of the fork might have a large wire delay. Our construction below introduces only a *local isochronic fork* between a pair of gates that are embedded within an SP process; all other wires are delay insensitive. This also shows that the isochronic fork requirement can be kept contained within a very small local (two gate) region without impacting the possible computations.

Note that a CMOS gate, or a CMOS gate combined with an inverter on its output (where we assume the inverter + gate combination can be viewed as a monolithic gate) can be described as a single SP process that has one internal state (i.e., no memory), and whose transition function $\hat{\delta}$ only depends on the values of its input variables. A collection of such gates with no internal connectivity (i.e., the output of each gate is an output of the SP process, and no output is used as an input to any of the gates) is also a simple SP process that has one internal state. These are the simplest SP processes because they do not have any local memory and only use their input variables to compute their output variables. We refer to such processes as ***SP-gates***.

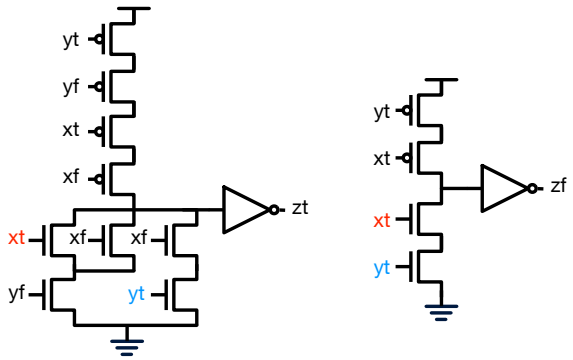

Fig. 1. A four-input two-output NAND2 process, with an internal isochronic fork between the two gates (signals `xt`, `yt` are inputs to both gates). The two inputs are `(xt,xf)` and `(yt,yf)`, and the output is `(zt,zf)`. Staticizers are not shown for clarity.

We begin by showing how arbitrary n-bit input, m-bit output functions can be implemented using a collection of two-output SP-gates. We do so by following the function block compilation strategy from [8].

Suppose the n input bits are $x_0, \ldots, x_{n-1}$, and the m output bits to be computed are $y_0, \ldots, y_{m-1}$.

- We can write m Boolean expressions $f_0, \ldots, f_{m-1}$ such that $y_i = f_i(x_0, \ldots, x_{n-1})$.
- Each input and output bit is encoded using a dual-rail code, with two wires per bit.
- We introduce m SP processes, each of which has $2n$ inputs corresponding to the dual-rail encoded x_i values, and produces a particular output y_j encoded as a dual-rail output. When the process receives a valid encoded input on all its inputs (n bits encoded using a dual-rail code on $2n$ wires), it computes its single encoded output bit and produces that on its output. If the process receives an all-zero input, it sets its two outputs to zero as well. This can be implemented with a single internal state, since the action taken by the SP process is completely determined by the value of its input variables.
- Since each dual-rail encoded input is needed by each of the m processes, a collection of fork processes are used to replicate the inputs.

This strategy can be used to implement *any* function with processes that have at most two outputs. Also, because two input NAND gates are universal primitives, we can also implement each f_i using a collection of processes, each of which have at most four inputs and two outputs where the collection simply implements the Boolean function using SP-gates that

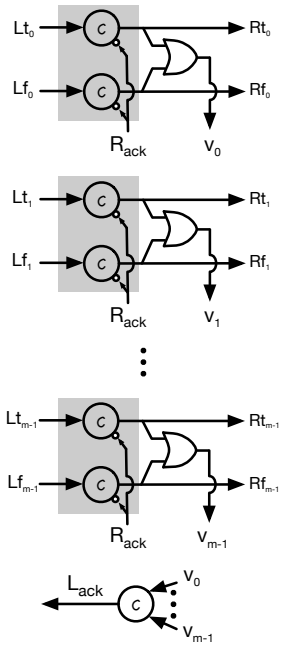

Fig. 2. Converting a collection of $2m$ dual-rail wires into a pipeline stage. The R_{ack} signal is the right acknowledge for the data output Rt_0, Rf_0, ..., Rf_{m-1}, and the L_{ack} signal is the left acknowledge signal. The input dual-rail wires for bits Lt_0, Lf_0, ..., Lf_{m-1} pass through a C-element to produce the output dual-rail wires. The large m-way C-element that generates L_{ack} is decomposed into a completion tree of two-input C-elements. The highlighted C-elements are grouped into processes.

implement the NAND function (the inputs are encoded using dual-rail codes, doubling the number of inputs). This means any function can be implemented by a collection of NAND processes—processes that have at most four inputs and two outputs, and where each process has a single state (i.e. zero bits of internal memory). The circuit implementation of the NAND process is shown in Figure 1.

Next, we can convert this into a pipeline stage. To do so, we need to introduce left and right acknowledge signals. A simple weak-conditioned pipeline stage can be designed by simply adding C-elements to each of the $2m$ output wires in the usual way, and then combining the $2m$ outputs using a completion tree to generate the acknowledge for the pipeline stage (see Figure 2). Each per-bit group of two C-elements that take three inputs (a dual-rail code and the right acknowledge R_{ack} signal) and produce two outputs (the dual rail output) are also grouped into a single signalling process; this process has the R_{ack} variable as an internal isochronic fork. The m copies of the R_{ack} signal needed by each of the m bits are generated using a standard fork process. All of the newly introduced processes are SP-gates that have at most three inputs and at most two outputs.

Since we can build a pipeline stage that can implement any function, it should be clear that we can build arbitrary computations using the construction described above. Hence, it is possible to implement any computation using forks and SP-gates with at most four inputs and at most two outputs. Thus, we conclude:

Theorem 4: A pipeline stage that computes any Boolean function can be implemented with a collection of SP-gates, each of which have at most four inputs and at most two outputs.

In particular, Theorem 4 means that limiting isochronic forks to be local to a pair of gates is sufficient to implement arbitrary computation.

VI. Discussion

We presented asynchronous signalling processes, a model for circuits in which modules are connected by delay-insensitive wires. Several models in the literature can be expressed within this framework. We studied the limitations of the computational power of signalling processes as a function of the number of outputs of processes.

We showed that processes with arbitrarily complex internal logic but single outputs are still extremely limited in terms of the input/output computations they can perform. We further showed that allowing processes to have two outputs is sufficient to enable arbitrary computations. In fact, Theorem 4 shows that two-output processes that are simply a pair of gates (that we call SP-gates) are enough to permit the design of pipelines that can implement arbitrary functions. Hence, only very simple two-output SP processes are necessary to enable general computation.

Our technical analysis offers an interesting insight as to why single-output asynchronous processes are extremely limited. Lemma 2 guarantees that between any two sets of inputs to the process, the output must signal at least once, while Theorem 3 implies that for all but a finite number of times in a computation, inputs rounds and the out signal must alternate. Since a wire can only signal by flipping its value, we obtain that, intuitively, the output signal can't provide nontrivial information about a value computed by the process. For a two-output process, this is no longer the case. The outputs can serve both for acknowledging that input signals have arrived, and to provide information about computed values.

There have been many proposals in the literature for a collection of modules that can be composed to build delay-insensitive circuits. All the proposals include multiple output components. Our results show that this is not a coincidence, because single-output asynchronous signalling processes are very limited—no matter how complex the module might be internally.

Keller proposed a set of modules that can be used to implement delay-insensitive computation. His proposal includes components with internal state as well as multiple outputs, such as the select primitive with three inputs, four outputs, and two internal states [9]. Ebergen developed a set of delay-insensitive components that were computationally general [2]. All his modules had at most three outputs. Ebergen also showed that eliminating the three-output module (the $RCEL$) from his components limits the generality of delay-insensitive traces that can be generated by the resulting circuits.

Other proposals for standard libraries of delay-insensitive components also included modules with multiple output wires. Another proposal includes components with five outputs and three inputs (the *D-call* primitive) [10]. Recent work proposed an optimized set of primitives for delay-insentive circuits that include modules that have three outputs such as the Mem primitive [11].

The asynchronous signalling processes framework that we introduced permits general processes, but the wires between processes are delay-insensitive. Thus, a limitation of asynchronous signalling processes is that they cannot model timing constraints on wires between modules. One common scenario where this occurs is in bundled-data communication protocols, where a bundle of data wires is assumed to have a lower delay than the corresponding request wire. We can "convert" such a circuit into one that is amenable to analysis by introducing a converter from/to the bundled-data protocol to/from a delay-insensitive communication protocol in the usual way—the sender would generate a dual-rail code from the single rail data wire combined with an inverter, two AND gates, and the request wire; the receiver would check that the dual-rail input is valid and generate a request signal using a completion tree. This modified circuit would be amenable to analysis using our framework.

In future work, we plan to investigate more general models of signalling processes. In particular, we plan to consider bounded delays in which an upper and lower delay bound is associated with each wire. With this extension, we will be able to directly model bundled data protocols. Furthermore, including this extension also enables the analysis of general timed asynchronous circuit families at the granularity of gates rather than processes. We believe that such an extension will lead to further insights into the properties of general asynchronous circuits.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for comments that improved the presentation of this paper.

REFERENCES

[1] W. A. Clark, "Macromodular computer systems," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pp. 335–336, ACM, 1967.

[2] J. C. Ebergen, "A formal approach to designing delay-insensitive circuits.," *Distributed Computing*, vol. 5, pp. 107–119, 12 1991.

[3] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *Proceedings of the 6th MIT conference on VLSI* (W. J. Dally, ed.), pp. 263–278, MIT Press, 1990.

[4] R. Manohar and Y. Moses, "The eventual C-element theorem for delay-insensitive asynchronous circuits," in *Asynchronous Circuits and Systems (ASYNC), 2017 23rd IEEE International Symposium on*, pp. 102–109, IEEE, 2017.

[5] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[6] R. Manohar and Y. Moses, "Analyzing isochronic forks with potential causality," in *Asynchronous Circuits and Systems (ASYNC), 2015 21st IEEE International Symposium on*, pp. 69–76, IEEE, 2015.

[7] R. Manohar and A. J. Martin, "Quasi delay-insensitive circuits are Turing-complete," in *Proceedings of the 2nd IEEE International Symposium on Asynchronous Circuits & Systems (ASYNC)*, 1996.

[8] A. J. Martin, "Asynchronous datapaths and the design of an asynchronous adder," *Formal Methods in System Design*, vol. 1, no. 1, pp. 117–137, 1992.

[9] R. M. Keller, "Towards a theory of universal speed-independent modules," *IEEE Transactions on Computers*, vol. 100, no. 1, pp. 21–33, 1974.

[10] P. Antognetti, P. Danielli, A. De Gloria, P. Faraboschi, and M. Oliveri, "A standard cell set for delay insensitive vlsi design," in *ASIC Conference and Exhibit, 1992., Proceedings of Fifth Annual IEEE International*, pp. 123–126, IEEE, 1992.

[11] P. Patra and D. S. Fussell, "Efficient building blocks for delay insensitive circuits," in *Proceedings of the First International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pp. 196–205, IEEE, 1994.

APPENDIX

We now present the proof of Lemma 1:

Proof: Suppose that $\langle p,t\rangle \prec_s \langle q,t'\rangle$. We first show that $t < t'$, by structural induction on the derivation of $\langle p,t\rangle \prec_s \langle q,t'\rangle$. If $\langle p,t\rangle \prec_s \langle q,t'\rangle$ is an instance of the Locality condition, then $t < t'$ by assumption. If it is obtained by the Successor condition, then $\langle p,t\rangle \hookrightarrow_s \langle q,t'\rangle$, and hence $t = t'-1 < t'$. Otherwise, $\langle p,t\rangle \prec_s \langle q,t'\rangle$ is derived via the Transitivity condition, i.e., for some computation r and time m, both $\langle p,t\rangle \prec_s \langle r,m\rangle$ and $\langle r,m+1\rangle \prec_s \langle q,t'\rangle$. By the inductive hypothesis we have that $t < m$ and $m+1 < t'$, and thus $t < t'$ as claimed.

Now suppose that $p \neq q$. We prove by induction on the derivation of $\langle p,t\rangle \prec_s \langle q,t'\rangle$ that if $\langle p,t\rangle \prec_s \langle q,t'\rangle$ then there is a signalling chain in s from $\langle p,t\rangle$ to $\langle q,t'\rangle$. Clearly, $\langle p,t\rangle \prec_s \langle q,t'\rangle$ cannot be derived by the Locality condition, because $p \neq q$. If it is obtained via the Successor condition, then $t' = t+1$ and $\langle p,t\rangle \hookrightarrow_s \langle q,t'\rangle$. In this case the signalling chain consists of the process singleton $\langle q\rangle$ (i.e., $k = 1$ and $x_1 = q$) and the singleton time sequence $\langle t\rangle$. Otherwise, $\langle p,t\rangle \prec_s \langle q,t'\rangle$ is derived via the Transitivity condition, i.e., for some computation r and time m, both $\langle p,t\rangle \prec_s \langle r,m\rangle$ and $\langle r,m+1\rangle \prec_s \langle q,t'\rangle$. If $r = p$ then by the inductive hypothesis there is a signalling chain between $\langle r,m+1\rangle$ and $\langle q,t'\rangle$. Since $t < m$, the same chain is also a signalling chain from $\langle p,t\rangle$ to $\langle q,t'\rangle$. Similarly, if $r = q$ then $m+1 < t'$ and the signalling chain from $\langle p,t\rangle$ to $\langle r,m\rangle$ is also a signalling chain from $\langle p,t\rangle$ to $\langle q,t'\rangle$. Finally, if $p \neq r \neq q$, then there is a signalling chain from $\langle p,t\rangle$ to $\langle r,m\rangle$ with with process sequence $\sigma = \langle x_1,\ldots,x_k\rangle$ and times $\langle t_1,\ldots,t_k\rangle$, and a chain from $\langle r,m+1\rangle$ to $\langle q,t'\rangle$ with process chain $\sigma' = \langle y_1,\ldots,y_\ell\rangle$ and time sequence $\langle t'_1,\ldots,t'_\ell\rangle$. Notice that $x_k = r$ by definition of signalling chains. Moreover, $t_k < m < m+1 \leq t'_1$, and so $t_k \leq t'_1 - 2$. It follows that the concatenation $\sigma \cdot \sigma' = \langle x_1,\ldots,x_k,y_1,\ldots,y_\ell\rangle$ of the two process sequences, together with the corresponding time sequence $\langle t_1,\ldots,t_k,t'_1,\ldots,t'_\ell\rangle$ define a signalling chain from $\langle p,t\rangle$ to $\langle q,t'\rangle$. The claim follows. □