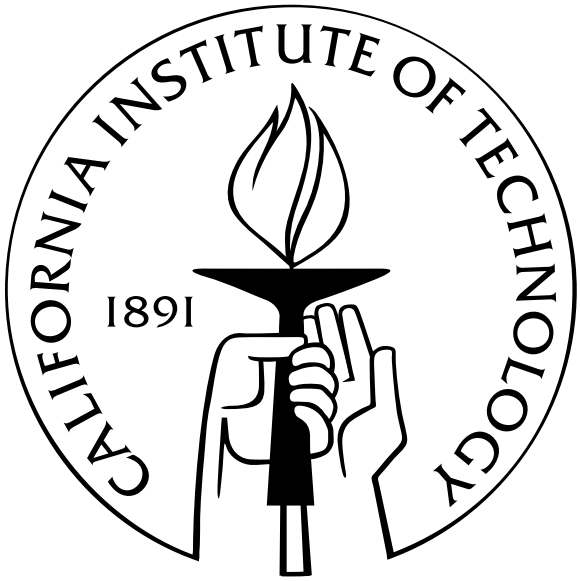


Design Rules for Non-Atomic Implementations of PRS

Karl Papadantonakis

Computer Science Department
California Institute of Technology

caltechCSTR/2005.001

Design Rules for Non-Atomic Implementations of PRS

Karl Papadantonakis
1/18/2005

Martin Synthesis yields quasi–delay-insensitive (QDI) circuits, expressed in production–rule-set (PRS) form. Under an atomic circuit evaluation model, these circuits are provably correct. However, not all physical circuit implementations provide the atomic transitions needed to satisfy the atomic circuit model. This can cause operational failures in real circuits, as we illustrate. Nonetheless, circuits with non-atomic transitions can faithfully implement the atomic circuit model when combined with a few simple slewtime constraints. To generalize this, we present a non-atomic circuit model, and we prove that any non-atomic circuit satisfying the slewtime constraints implements the atomic circuit model. To synthesize correct physical circuits, therefore, one can use Martin Synthesis assuming atomicity, and then physically implement the resulting circuit using the slewtime constraints as design rules.

0.1 Acknowledgements

Thanks to Alain Martin for developing the foundations of the subject, for an elegant P/N fork model, for suggesting that I consider a multiatomic model, and for highlighting the question of whether variable-assignments are assumed to be atomic. Thanks to Mika Nyström for writing the `alint` program, for experimentally demonstrating that slewtime constraints can be used as design rules, for suggesting that I justify the introduction of global time, and for words like "diatomic". Thanks to André DeHon for help with the abstract and general work partitioning. Thanks to the Caltech Asynchronous VLSI Group for feedback and for developing the background material.

Contents

1 Introduction

Martin Synthesis is a methodology used to construct an asynchronous circuit from a specification. The resulting circuit, expressed in Production Rule Set (PRS) form, is Quasi–Delay-Insensitive (QDI), i.e. it is **correct** in the absence of assumptions about operator delays[1]. We shall assume that **correctness** means there is a proof that *all possible circuit behaviors* adhere to the specification.

To prove anything about *all possible circuit behaviors*, it is important to have a definition of what constitutes *all possible circuit behaviors* for any PRS (or, at least, for any synthesizeable PRS). Such a definition is called a **semantic model**.

We argue that the most important attribute of the semantic model of PRS is whether or not transitions are *atomic*. This attribute was previously thought to be irrelevant[2][7]. In any semantic model of PRS, it is necessary to define *events* and to restrict the ordering between some of them. Transitions are **atomic** if, in each behavior, each transition is a *single event*.

We begin the paper by reviewing the formulation of correctness used in Martin Synthesis (section 2). We show that if transitions are atomic, then the PRS is correct without additional assumptions (section 3.4). However, if the transitions are non-atomic, then slewtime assumptions are required (section 3.3). The rest of the paper is devoted to proving the sufficiency of these slewtime assumptions.

We begin our formulation by presenting **observation**, (section 3.5) a method of understanding when the non-atomic behaviors implement an atomic specification. We then justify a choice of non-atomic model (section 3). We proceed to formalize the atomic model, (section 4) the non-atomic model (section 5), observation (section 6), and the necessary slewtime constraints (section 7). Finally, we prove that these constraints are sufficient (section 8), and consider the practical implications of this result (section 9).

2 Background on Martin Synthesis

Martin Synthesis is a method for compiling high-level specifications into QDI circuits. The target language describing the resulting circuits is called Production Rule Sets (PRS) [1].

The circuits compiled by Martin Synthesis are entirely asynchronous and do not use a clock. These circuits have been used in several high-performance CPU designs, including the CAM[9], MiniMIPS[3], and Lutonium[10].

2.1 Correctness by Construction

Martin Synthesis is correct by construction. This means that the construction method generates a proof that all possible dynamic behaviors of the final design adhere to the high-level specification that the engineer starts with. The design is initially specified as a sequential program and transformed through a sequence of description levels (CHP, HSE, and PRS). Each description in the sequence provably implements its specification, the previous description. The final design is a chip implementing the original specification.

2.2 Quasi–Delay-Insensitivity (QDI)

Delay insensitivity (DI) means delay can be added to any operator or wire without affecting correctness. This allows more transformations to be applied than what would otherwise be possible. Unfortunately, strict DI can only implement trivial specifications.[2]

Quasi–Delay-Insensitivity (QDI) allows non-trivial specifications to be implemented by allowing delay on *operators* but not on *wires*. This property holds for all behavior models of all description levels used in Martin Synthesis (CHP, HSE, and PRS). The properties hold because these models *only* assume that commands execute *eventually* when they are enabled. In particular, there are no **timing assumptions**. I.e. the models do not allow assumptions of the form "command A completes before command B because command A is faster than command B". Commands can wait for other commands to complete, but they cannot rely on the speed of other commands:

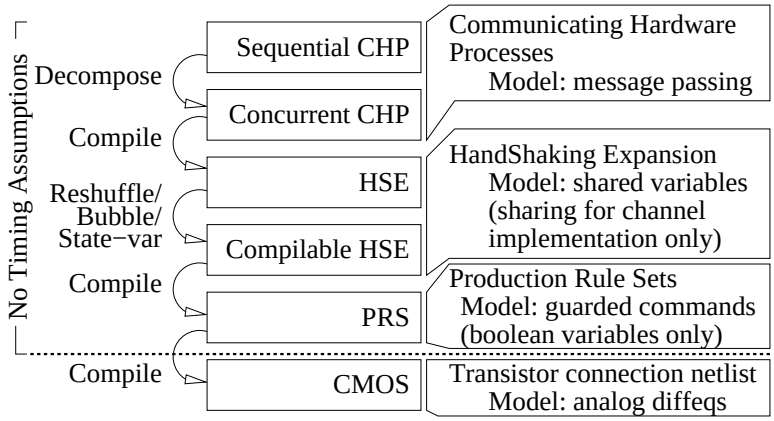


Figure 1: *Martin Synthesis: No Timing Assumptions.*

In this paper, we will show that timing assumptions must be added in the final step of compiling PRS into CMOS circuits. However, we will use an existing abstraction of PRS, so as not to require timing assumptions to be added to any of the higher-level compilation steps.

2.3 Production Rule Sets (PRS)

A PRS is a set of nodes with initial values (0 or 1), and a set of gates connecting to those nodes. In the simplest version of PRS, each circuit gate, with output node y, is expressed as exactly two PRs, $g \rightarrow y\downarrow$ and $q \rightarrow y\uparrow$. g and q are boolean functions, called **guards**, of other nodes in the circuit. The guard g determines when the pulldown network (PDN) conducts, and the **opposing guard** q determines when the pullup network (PUN) conducts:

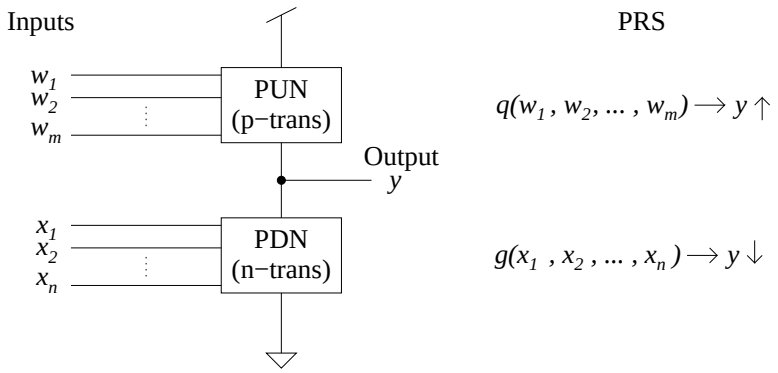


Figure 2: *The CMOS-gate implementation of a general pair of PRs.*

In Martin Synthesis, PUNs are implemented only in p-transistors, and PDNs are implemented only in n-transistors, as this results in the best voltage regeneration.

As shown in Figure 3, PRS describes both combinational gates such as NAND gates, and state-holding gates such as C-elements. If the guards are complementary, (as in the NAND gate) then for any input, exactly one network conducts and so the output is always a function of the current input; i.e. the gate is combinational. Otherwise there is an input combination for which neither network conducts, and it is assumed that in this case the gate holds state.

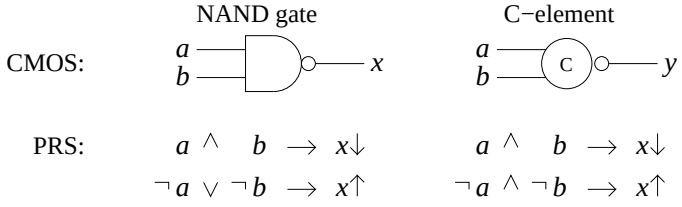


Figure 3: *PRS for NAND and C-element gates.*

Interpreting $y\downarrow$ and $y\uparrow$ as shorthand for $y :=$ **false** (i.e. 0) and $y :=$ **true** (i.e. 1), respectively, a generic production rule then has the form:

$$g \rightarrow y := v$$

where g is the guard, y is the **target node**, and v is the **target value**.

2.4 Semantics of PRS

There are a number of possible semantics for PRS. Before discussing a particular semantic model, let us recall[1] three important properties that any PRS semantics should have:

1. An execution is a set of **events**. Each event is the firing of a PR. The firings of a single PR form a sequence. In other words, an execution is some sort of **concurrent composition**[1] of PR-firing–sequences. (In this paper, we will henceforth avoid using the phrase "concurrent composition", however, as it is a vague term that does not indicate whether or not sequences are interleaved atomically. See section 3.4).

2. The **system state** is an assignment of a value to each node. The effect of an event is to change the value of the target node of the PR to the target value of the PR. I.e. after the event executes the system state is the same as it was before the event executed, except for this change.

3. An execution must satisfy liveness and safety. A PR is (nonvacuously) **enabled** whenever its guard is true and its target does not yet have its target value. **Liveness** (a.k.a., **progress**) is the requirement that any enabled PR eventually fire (or become disabled). **Safety** is the requirement that an event can only occur when its PR is enabled. (To simplify analysis, we have ruled out ineffective and vacuous events. An equivalent approach[1] would be to allow such events but to ignore them in analysis).

2.5 Stability and Noninterference

Any semantics should have the notions of stability and noninterference[1]:

1. An execution is **stable** if: a PR is never disabled, except through the firing of that PR.

2. An execution is **noninterfering** if for each gate with guard g and opposing guard q, $(g \wedge q)$ holds always.

We assume that stability and noninterference are part of the specification of every PRS, since our main result depends on these properties, and they are guaranteed by Martin Synthesis.

2.6 Martin Synthesis Example: $*[L; R]$ Buffer

After decomposition, a design is expressed as a set of CHP processes, typically of the "buffer" form $*[L; R]$. Such a process repeatedly receives data on input channel L and sends a function of that data (such as an arithmetic operation) on output channel R. For now, ignore the fact that data is sent; we will consider a circuit with the proper communication sequence, and data can easily be added later. We are using the following notation:[1]

Expression	**Meaning**
L	communication on channel L
L Le	wires implementing channel L (unless otherwise noted, initial wire value is 0).
$[Re \wedge L]$	Wait for condition $[Re \wedge L]$ to hold.
$*[$`Body`$]$	Repeat `Body` forever.
$S_1; S_2$	Perform S_1 and then S_2, sequentially.

Figure 4: Expressions used in CHP and HSE

Each CHP process is compiled into HSE as follows. First, each communication on a channel is compiled into (i.e. replaced by) a sequence of **handshake phases** on the wires implementing that channel. Assuming four-phase handshakes, L is compiled into $Le\uparrow; [L]; Le\downarrow; [\neg L]$, and R is compiled using a complementary handshake:

$$
\begin{array}{l}
\text{CHP Process} \qquad *[\, L \qquad\qquad\qquad ; R \qquad\qquad\qquad] \\
\\
\downarrow \text{ compile (communications} \rightarrow \text{DI handshakes)} \\
\\
\text{HSE Process} \qquad *[\, Le\uparrow; [L]; Le\downarrow; [\neg L]\, ; [Re]; R\uparrow; [\neg Re]; R\downarrow \quad] \\
\\
\downarrow \text{ reshuffle} \\
\\
\text{HSE Process } \{Le\uparrow\}; *[\;\; [Re \wedge L]; R\uparrow; Le\downarrow; [\neg Re \wedge \neg L]; R\downarrow; Le\uparrow \;\;] \\
\\
\downarrow \text{ add bubble and state variables} \\
\\
\begin{array}{l}\text{Implementable} \\ \text{HSE Process}\end{array} \qquad \ldots; *\,[\,[Re \wedge L]; \_R\downarrow; Le\downarrow; [\neg Re \wedge \neg L]; \_R\uparrow; Le\uparrow\,]
\end{array}
$$

Figure 5: *Using Martin Synthesis to obtain implementable HSE for the* $*[L; R]$ *buffer.*

In **reshuffling**, the phases are then re-ordered to form a new interleaving of the L and R handshakes which is easier to implement. Additional variables are then added to facilitate implementation. In the above example, just one variable was added: $\_R$ (of which R becomes a negated copy). The HSE is now ready to be implemented in PRS.

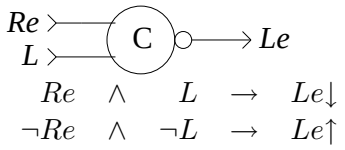
The last step is to find a PRS that implements the following HSE:

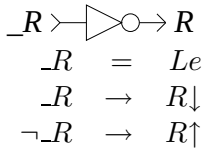
$$*[[Re \wedge L]; \_R\downarrow; Le\downarrow; [\neg Re \wedge \neg L]; \_R\uparrow; Le\uparrow] \tag{1}$$

Such a PRS can be found using the following procedure[1]:

1. Assume the hypothesis that the HSE holds.
   Compute the values of all output variables at each semicolon, using this assumption.
2. Ensure that when a PR's assignment appears in the HSE, the PR is enabled.
3. Ensure that no PR having the form $\cdots \rightarrow y := v$ is ever enabled when $y \neq v$ in the HSE.

The simplest PRS obtainable in this manner for our HSE specification is as follows:





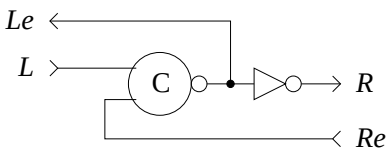


Figure 6: *Martin-Synthesized PRS for the* $*[L; R]$ *buffer derived in Figure 5.*

3 The Question of Atomicity

3.1 Feedback Example

The PRS buffer implementation shown in Figure 5 makes assumptions about its environment. For example, it assumes that each transition on R will be followed by a single transition in the opposite direction on Re. This R/Re environment can be modeled by an inverter. By bringing this inverter into the circuit, we can remove all actions on R and Re from the HSE, obtaining the following HSE and circuit:

$$*[[L]; Le\downarrow; [\neg L]; Le\uparrow]$$

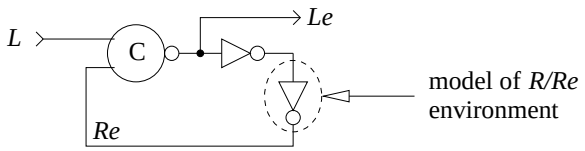


Figure 7: *Circuit vulnerable to slow transitions on* x.

To be correct, the circuit must behave according to the HSE. In this case, the HSE predicts one output transition on L per input transition on Le (assuming the environment waits for each L transition before producing an Le transition). We now proceed to show that the HSE is violated for slow transitions on L.

3.2 Spurious Ring Oscillator

The analog CMOS implementation of the circuit shown in Figure 7 is shown at left in the following figure (we have renamed the nodes without changing the circuit):

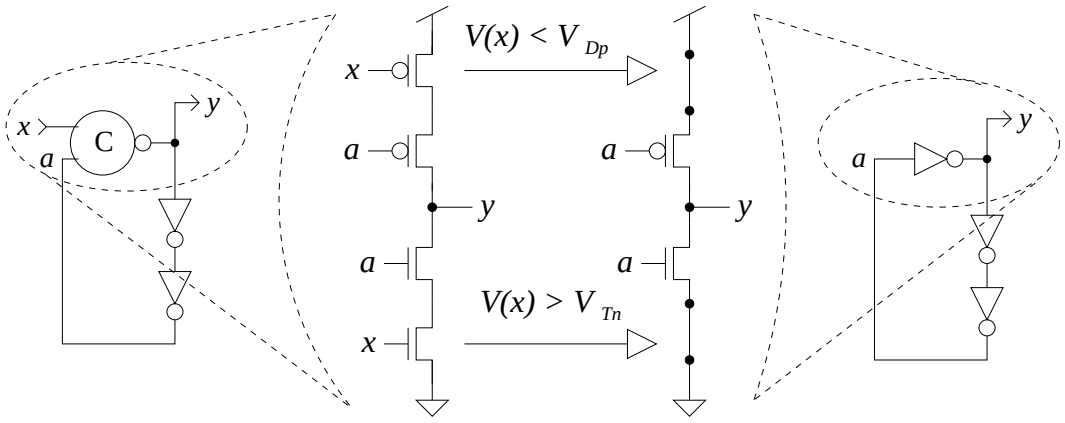


Figure 8: *When x has an intermediate analog value, it enables N and P transistors simultaneously. This leads to behavior not possible in a digital, atomic model.*

Suppose x rises very slowly, so that for a long period of time its analog value is simultaneously one n-threshold (i.e., one V_{Tn}) above ground and one p-threshold (i.e., one V_{Tp}) below the supply voltage V_{DD}. If the C-element in is implemented as shown above, then it behaves as an inverter when x has the intermediate analog value.

This virtual inverter is in a loop with the other two inverters, forming a free-running ring oscillator. This results in failure: for each input transition, there are an arbitrary number of output transitions, while an atomic model of the PRS for Figure 7 predicts just one.

3.3 Necessity of Slewtime Restrictions

We define a **transition** as the time interval when $V_{Dp} < V(x) < V_{Tn}$, and the **slewtime** as the duration of the transition. The previous example illustrates that slewtimes must be restricted to achieve specified behavior.

This fact has long been known. The most obvious possible problem with unrestricted slewtime is that two transitions on a single *node* might overlap in time. This problem is called a **transmission error**, and can be avoided by requiring slewtimes to be less than feedback delays, as shown by Alessandro DeGloria et. al.[5]

A more general problem with unrestricted slewtime is that a single transition might overlap with a sequence of transitions on a single gate input, as in the example in section 3.2. This problem is coincidentally avoided by checking the same condition (slewtimes must be less than feedback delays). This fact has also been known for some time, though only experimentally. Spurious oscillators were avoided in the MiniMIPS project by checking slewtimes using the `alint` program[4].

In this paper we will prove in general that any PRS resulting from Martin Synthesis properly implements its HSE if slewtimes are restricted.

3.4 Theoretical Solution: Atomic Assignments

We can rule out the unwanted behaviors by assuming that assignments to variables are atomic. I.e. we must assume that each firing is a *single action in the execution*. Whether or not the circuit evaluation model is atomic depends on what types of interleavings of PR firing sequences are allowed. The simplest atomic model defines an execution as any *sequence* of PRs satisfying safety and progress. Martin Synthesis is correct in this model[1].

Martin Synthesis is in fact provably correct in any atomic model. We show that any atomic model is equivalent to the sequential model, as follows: first we show that the sequential model is equivalent to a timestamp model (section 4.2). Then we show that any atomic model can be embedded in global time (section 11).

3.5 Practical Solution: Observation

Unfortunately, a model of atomic assignments is not physical. It is equivalent to zero slewtime, which is not always directly implementable. We therefore must allow an underlying non-atomic model. Since the non-atomic model does not always behave properly, however, we must determine constraints such that this model *implements* the atomic model.

What does it mean for one model to implement another? The atomic model is implemented if for any non-atomic execution L, the results can (at the very least) be *observed* on an oscilloscope and they agree with some atomic execution. If the atomic model said the program would output the number 13, then the final implementation (the chip) ought to output the number 13. We will formalize observation in section 6.

3.6 Our Non-Atomic Model: The Diatomic Model

A non-atomic model must define the effect of a non-instantaneous transition. Consider a downgoing transition (i.e. an assignment of value 0 to a variable that was previously 1). Before the assignment, the variable has value 1, and after the assignment, the variable has value 0. So far, we have not described anything that doesn't happen in an atomic assignment.

The fact that the assignment is non-atomic means that there are local effects *during* the assignment which cannot be modeled as a single action on the binary variable. There are several effects worth considering:

1. **P/N Transistor Thresholds**. As in Figure 8, the implementation of the variable might be in an analog region where it simultaneously enables both pullups and pulldowns.

2. **Isochronic Fork**. The variable might consist of an isochronic fork[1]: there might be copies of the variable which are assumed to switch at the same time. However, there will always be some difference in which the variable is seen to switch at the different copies. While some copies have received the new value but others have not, the variable might simultaneously enable pullups and pulldowns, just as in the P/N transistor thresholds case. We analyse this model in section 10.1.

3. **Non-Monotonic**. Due to noise, the variable implementation might cross a logic threshold several times[1].

All of the above examples can be understood in terms of a 0-X-1 model: after an assignment, the variable has the assigned digital value until a subsequent assignment begins. *During* an assignment, the variable has the value "X", which might enable any (otherwise-enabled) production rule in which it appears. This is the weakest possible assumption:

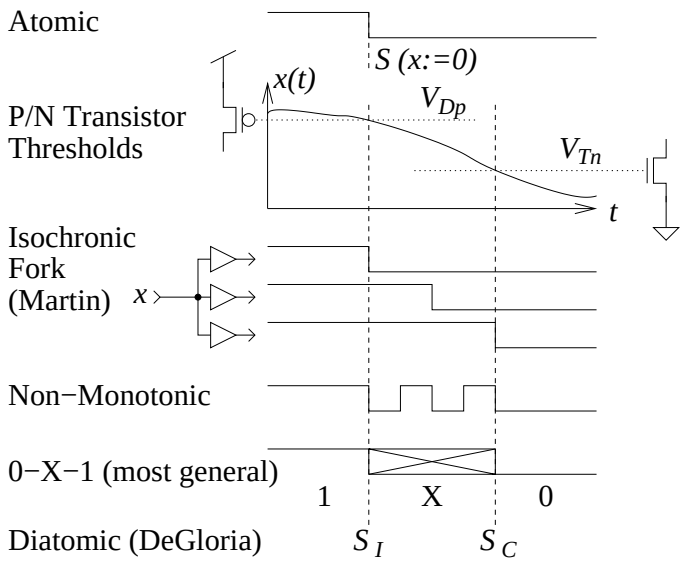


Figure 9: *A wide variety of non-atomic models can be understood as a diatomic 0-X-1 model.*

Also, all of the above models can be analysed as what we call a **diatomic** model: a model in which each assignment S is executed as two statements S_I and S_C which **Initiate and Complete** the assignment[5]. A diatomic model was previously used to analyse transmission errors, but not 0-X-1 states[5]. We extend the analysis to 0-X-1 states.

3.7 Diatomic Model Semantics: The P/N Fork Model

We model 0-X-1 behavior for any PRS, as follows. We construct a modified circuit in which each node x is extended by delayed nodes x_n and x_p. x_p feeds p-transistors; x_n feeds n-transistors. The atomic behaviors of the modified circuit are 0-X-1 behaviors. An assignment to x is considered to be in progress (i.e. $x = \mathrm{X}$) whenever $x_n \neq x_p$. As desired, x can simultaneously enabled pullups and pulldowns when in this state.

For example, suppose we want to verify the PRS system of Figure 7 in the P/N fork model. We would have to verify the atomic behaviors of the following circuit:

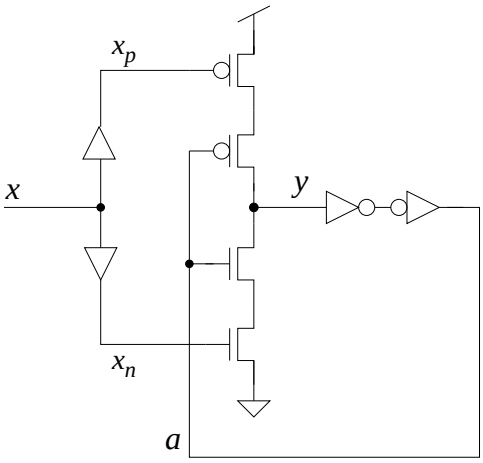


Figure 10: *System of Figure 7 in the P/N fork model*

This model is further discussed in section 5.1.

3.8 Global Time

Consider the following atomic behavior of the circuit shown in Figure 10:

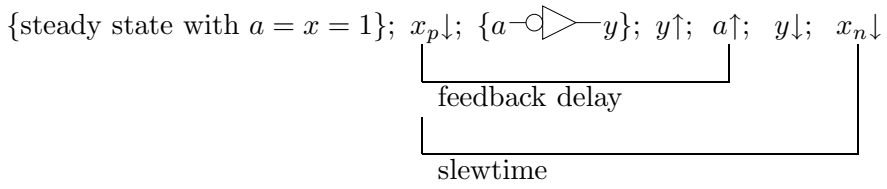


Figure 11: *Trace in which unsafe event $y\downarrow$ occurs.*

(the notation {a—▷—y} means that unless there are further x_p or x_n changes, the node y always eventually becomes $\neg a$).

The circuit fails when the unsafe event $y\downarrow$ occurs. The unsafe event is allowed to occur because $a\uparrow$ occurs before $x_n\downarrow$. Evidently, circuit correctness depends on the ordering of $a\uparrow$ and $x_n\downarrow$. This ordering is determined by *delays* along *different paths* through the circuit.

The most general way to model orderings caused by delays along different paths through the circuit is by introducing **global time**. I.e., we assume that each event occurs at some global time. This assumption allows comparison of delays along different paths. There is no loss of generality for making this assumption (see section 11. Also, the assumption of global time leads to the simplest possible model formulation.

4 Atomic Model Formulations

4.1 The Sequential Model

What are the possible semantic models of PRS? Recall that the firings of a single PR form a sequence. Thus, in any model, each firing of a PR must be distinguished from subsequent firings of the same PR. Therefore an event is generally a PR together with an index, timestamp, or other distinguisher. The simplest possibility is to define an execution as any *sequence* of PRs [1] satisfying liveness and safety (as defined in Section 2.4). Clearly this model satisfies the criteria: the firings of a single PR are distingushed by their increasing positions in the complete execution sequence, and form a subsequence. This model is called the **sequential model**.

Two important verification methods work in the sequential model:

1. One can prove that a PRS implements a sequential HSE by tabulating the state space[1].
2. Stable PRS is deterministic, so that if an arbitrary execution is correct, then all executions are correct, assuming the specification is limited to handshake sequences[8].

Even though these methods might be model-independent (within limits), we assume that when a PRS has been correctly constructed it is (at least) correct in the sequential model. The methods can at least be proven in this case.

4.2 The Timestamp Model and its Sequential Equivalence

We have just argued that the sequential model is a convenient abstract specification. However, it has the annoying restriction that events cannot occur simultaneously, while this could theoretically happen in a physical implementation. This discrepancy would complicate our analysis by preventing a direct correspondence between time in the specification and in the physical implementation.

The simplest possible time correspondence is achieved by using a specification model in which each event has a physical timestamp. Events are simultaneous if their timestamps have the same value. For consistency of our analysis we formulate a particular model having such timestamps, which we call the **timestamp model**. We complete its definition, to achieve the three necessary properties of PRS semantics, as follows:

1. An event is a PR together with a real number timestamp. To ensure that the firings of a single PR form a sequence, we consider only **calendars**: sets of events whose timestamps do not have limits other than infinity (see section 11.6), without simultaneous assignments to a single node.

2. System state: To simplify the safety condition, we adopt the following **endpoint convention**: a value assigned at time t holds over some semi-open interval $(t, t']$. Specifically, a node's state at time t is the target value of the latest event targeting the node *before time* t (or the initial value of the node if no such event).

3. An execution is any live, safe calendar. Liveness: no rule is indefinitely enabled. Safety: an event with timestamp t must be enabled *at time* t.

Under stability, the timestamp model implements the sequential model[1], and this is easily seen: the timestamp model does not eliminate any behaviors, but it introduces new behaviors in which there are groups of simultaneously firing PRs. Nonetheless, each of these groups can be exploded into a short sequence, i.e. a sequence that is fast enough that it does not overlap with other events in the execution. Under stability, this results in an execution without concurrently firing PRs.

Thus any timestamp execution implements a sequential execution. And the timestamp model is more convenient to physically implement than the sequential model. Therefore we choose the timestamp model as our atomic model (i.e., as our specification model).

5 Nonatomic Model Formulation

The goal of this paper is to understand when nonatomicity of PR firings causes circuit failure, so that conditions can be derived that are provably sufficient to eliminate all failures. To this end, we formulate a model which accurately describes the effect of nonatomic PR firings.

5.1 The P/N-Fork Model: A Diatomic Model

Nonatomic assignments arise in the CMOS implementation of a PRS. In the implementation, each node x connects to several transistor inputs. In particular, it is an input to at least one PDN and at least one PUN (see section 2.3). In general, the PDN input threshold V_{Tn} differs from the PUN input threshold V_{Dp} ($\stackrel{\text{def}}{=} V_{DD} - |V_{Tp}|$). The CMOS implementation of the circuit in Figure 7 can fail because the analog signal $x(t)$ is continuous and cannot cross both of these thresholds at the same time. A failure occurs if $x(t)$ spends too long in transition, i.e. in a region where it has crossed one threshold but not the other. Specifically, during the transition, x can enable both PUNs and PDNs. We will also consider the case where x enables neither PUNs nor PDNs, which occurs if $V_{DD} < V_{Tn+p}$ ($\stackrel{\text{def}}{=} |V_{Tn}| + |V_{Tp}|$).

Our model must describe the effect of a signal crossing one threshold without crossing the other. We do this as follows (thanks to Alain Martin for this suggestion): replace each circuit node x by two nodes, x_p and x_n. x_n is **true** whenever x enables PDNs, and x_p is **false** whenever x enables PUNs. A transition on x is some ordering of the same type of transitions on x_p and x_n. I.e., the nonatomic behavior of the original circuit is equivalent to the atomic behavior of a modified circuit in which x has two delayed copies, one feeding to PUNs and another feeding to PDNs:

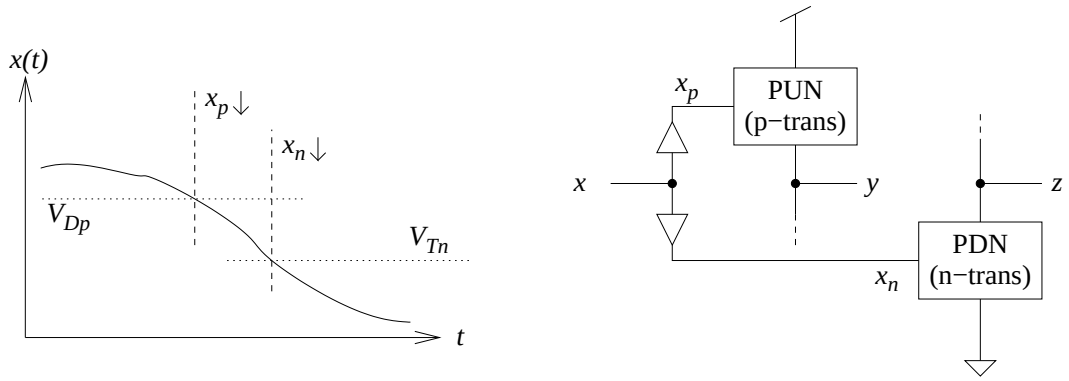


Figure 12: *The nonatomic transition $x\downarrow$ is equivalent to some interleaving of $x_p\downarrow, x_n\downarrow$ in the modified, atomically modelled circuit.*

We call this model the **P/N-Fork model**. It has the virtue of covering both interleavings of $x_p\downarrow, x_n\downarrow$, allowing us to change V_{DD} without changing our basic model.

5.2 Weak Vacuity Condition in the P/N-Fork Model

We will make the P/N-fork model into an execution model when we justify a choice of event format in Section 6.4. For now, assume we have an event format. As in the timestamp model, a calendar is an execution if and only if it satisfies safety (only enabled rules execute) and progress (enabled rules eventually are disabled). What remains is to define what it means for a rule to be (nonvacuously) enabled.

We define the system state just as in the timestamp model, except now we assign values to the new variables of the form x_n and x_p. A pulldown guard is evaluated in terms of the n-subscripted versions of its variables, while a pullup uses the p-subscripted versions.

Recall that a rule is enabled if the guard is enabled *and the target has not yet been assigned*. This would be an ambiguous definition of enabling in the P/N-Fork model, since the target is now represented by two variables. We solve this problem by requiring that *both* variables differ from a new target value before that target value can be assigned. This choice will simplify our analysis by removing vacuous events from consideration.

5.3 Extending Stability and Noninterference to the Diatomic Model

In the atomic model, a PRS is stable if each production rule remains enabled until it executes. In other words, for each rule $g \to y := v$, the guard g can only be invalidated when $y = v$.

In the P/N-Fork model, y is implemented as two variables, and we say that a *diatomic* execution is **stable** if *both* of these variables must be assigned before g can be invalidated. In general, an atomically stable PRS is not diatomically stable. However, diatomic stability is important to the implementability of diatomic PRS, as it is a step towards guaranteeing analog stability. Diatomic stability implies (but is not guaranteed by) the absence of transmission errors (which have been separately analysed[5]). We therefore present diatomic stability as a low-level specification to the electrical implementation.

Similarly, atomic noninterference does not imply diatomic noninterference, so diatomic noninterference must also be proved. In summary, to prove correctness of diatomic PRS, we must show *three* properties:

1. The diatomic behaviors implement the high-level atomic specification (see section 6).
2. The diatomic behaviors are stable, adhering to the low-level specification.
3. The diatomic behaviors are noninterfering, adhering to the low-level specification.

A violation of property 1 is called an **execution error**, and a violation of property 2 or 3 is called a **transient error**. The remainder of this paper focuses on ruling out execution errors and transient errors.

6 Relating the Models using Observation

The primary goal of this paper is to prove that under slewtime constraints, nonatomic PRS implementations faithfully implement the atomic model. As a first step towards mathematical formulation of this statement, we have justified the choices of P/N-Fork and timestamp models as our nonatomic and atomic models, respectively. To complete the formulation, we now precisely define what it means for one model to implement another. For reasons soon to be stated, we will call this formulation the **observation theorem**.

6.1 Observation Rule and Observation Theorem Statement

In any design methodology, a circuit can only be considered correct if it is possible to view the output nodes of the circuit on an oscilloscope and **observe** the values given by the circuit's specification. We assume that *every* transition in the atomic model can be observed in the nonatomic implementation. This assumption simplifies analysis. There is no loss of generality in making this assumption, because at the level of PRS models, output nodes are not special. Output nodes can have the same functionality as any other nodes, which demonstrates that a node's functionality is not reduced by the fact that it can be observed.

In general when one designs hardware (for example, using Martin Synthesis) the behaviors of an implementation might contain more events (i.e., more PR firings) than the specification. It is also possible for one event to implement several specification events (usually a special optimization).

Fortunately, at the level of PRS models, we can avoid these complications by defining the implementation events in such a way that *every* implementation event corresponds to a unique specification event, with no two implementation events corresponding to the same specification event. We call this bijective correspondence an **observation rule**:

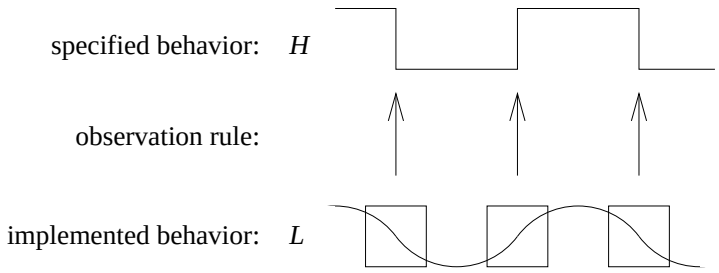


Figure 13: *L implements H through an observation rule.*

We can state the observation theorem in terms of any such observation rule:

Theorem 1 (Observation) *The following two properties hold for any diatomic execution L of any stable, noninterfering PRS satisfying slewtime constraints (see section 8.2 for formalization) and having no self-invalidating gates (see section 6.11):*

1. *L contains no transient errors.*
2. *Let H denote the calendar (in the specification model) consisting of the image of the observation rule when applied to L. H is an execution.*

Statement 2 says that there are no execution errors. To make it precise, we proceed to define our implementation events (subsection 6.4) and to select an observation rule (subsection 6.5).

6.2 Observation Theorem: Assumptions & Results

We now review the path that the mathematical objects take through the observation theorem when it is applied. Begin with a **CHP Specification**. It is compiled to **Atomic PRS** through Martin Synthesis. Slewtime constraints are added, giving **Diatomic PRS**. We pick an execution, L. Applying the observation rule ϕ, we obtain the atomic calendar H. The observation theorem is allowed to modify H producing atomic executions H' (as many times as it wants to). By Martin Synthesis, these H' satisfy (atomic) stability and noninterference. The observation theorem then proves the execution properties of (atomic) safety and progress; i.e. it proves that H **is an execution**. It also proves transient properties: that L satisfies **(nonatomic) stability and noninterference**.

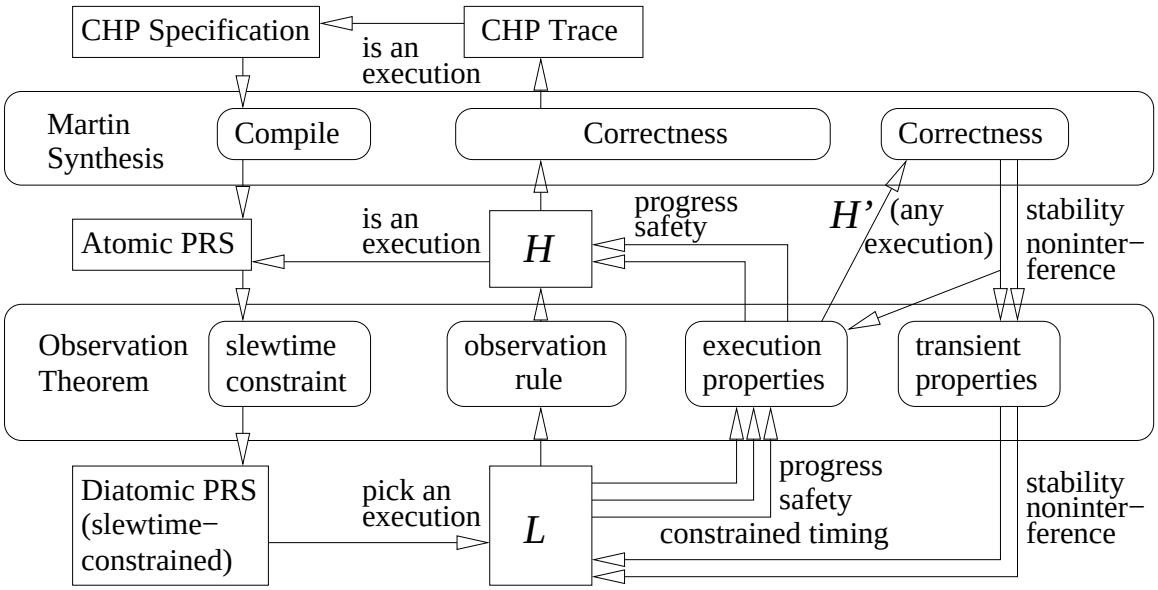


Figure 14: *Assumptions & Results of the Observation Theorem*

6.3 Observation Theorem: Proof Roadmap

We have four things to prove for any (constrained) diatomic L:

1. Its (atomic) observation H satisfies safety.
2. H satisfies (atomic) progress.
3. L is (diatomically) stable.
4. L is (diatomically) non-interfering.

We begin with the **Signal Lemma**, (sections 6.11 and 8.1) which relates the state at the same time in L and H. This gives us result 2, progress: if a rule is indefinitely enabled in H, then it must also be indefinitely enabled in L.

To prove the remaining facts, we partition the problem according to the ordering of S_D, S_E. In CMOS, this ordering is determined by the choice of the supply voltage V_{DD} (section 6.8). We begin by proving the result for low-V_{DD} executions (section 6.11), i.e. ones in which S_E happens first.

We also argue that this case can be integrated into the high-V_{DD} case. We prove two lemmas, the **more-enabled execution lemma** and the **more-enabled transient lemma**, (section 6.11) which show that if we have proved the high-V_{DD} case, then the observation theorem holds for executions with no restriction on S_D, S_E ordering (i.e., both $S_D; S_E$ and $S_E; S_D$ can occur in the same execution).

Finally, we prove the high-V_{DD} case (section 8). Since we have already proved progress, there are three types of errors that must be ruled out:

1. unsafe event (section 8.3): see below
2. diatomic interference (section 8.4): similar argument to unsafe event
3. diatomic instability (section 8.5): transform to unsafe event

The archetypal example is the case of the unsafe event. Suppose (for contradiction) there is an unsafe event. Consider the earliest one, (r, t_E, t_D). We can assume all theorem results hold up to time t_E. The guard g (of rule r) is enabled in L but not in H. By the signal lemma, some terms of g are X. Now we note that if we could re-interpret the 'X's in H (e.g. imagine we could slightly change the observation rule) as 0 or 1, then we could enable g in H. We re-interpret the 'X's by attempting to **perturb** (section 7.4) the assignments to the variables. Note that we have to move them less than one slewtime. If we can perturb the 'X' terms without disturbing the other terms, then we can create a pulse in g. This contradicts stability.

6.4 Calendars in the P/N-Fork Model

Before we can state our observation rule, we need to select a mathematical structure to represent event "sequences" in the P/N-Fork model. For the same reasons that we selected the timestamp model over the sequence model as our specification model, we choose to distinguish events in the P/N-Fork model using timestamps.

Each assignment to x in the atomic model is implemented as two assignments in the P/N-Fork model: one to x_n and one to x_p. Each of these assignments has a timestamp. However, to satisfy our requirement (from the preceding subsection) that no two events implement the same specification event, we consider both of these timestamps as part of the same event. In summary, an event is a tuple of the following quantities:

1. r, the rule $g \to y := v$ that was executed.
2. t_n, the time at which y_n was assigned.
3. t_p, the time at which y_p was assigned.

As in the atomic timestamp model, a **calendar** is any set of events in which the timestamps form a limitless set, and different simultaneous assignments are never to the same node. Also as in the atomic case, an **execution** is any live, safe calendar. We use the definition of enabledness developed in Section 5.2. To make the model easiest to implement, we only require (for safety) that each event be enabled at the earlier of t_n or t_p.

At first sight, this choice of enabledness might appear to complicate analysis. However, the following theorem, based on an argument combining nonatomic stability and continuity, shows that our analysis can assume the event is enabled at both times:

Theorem 2 (Nonatomic Enabledness) *If the observation theorem can be proved assuming that each event (r, t_n, t_p) in a nonatomic execution is enabled both at time t_n and at time t_p, then the theorem also holds if it is only assumed that such an event is enabled at time* $\min(t_n, t_p)$.

Suppose the theorem did not hold without the assumption. Then there is some execution L in which the guard is satisfied at t_n but not at t_p, with $t_n < t_p$ (without loss of generality). Consider such case(s) with earliest t_p. Create a new execution by moving the sub-assignment S_p earlier, to time t_n. By assumption, the proven theorem now holds up to time $t_n + \epsilon$. By nonatomic stability, S_p is still enabled after time t_n at a new time $t_n + \epsilon'$, for some $\epsilon' \leq \epsilon$. We continue delaying S_p and applying the proven theorem until S_p reaches t_p, its original time, showing that the guard was satisfied at time t_p. □

6.5 Choice of Observation Rule

In the nonatomic implementation, each assignment event (r, t_n, t_p) spans the time interval between t_n and t_p. To have an observation theorem, we need to map each such time interval to a single, atomic timestamp. There are many workable possibilities, since the atomic timestamps can be globally rescaled without affecting the logical correctness of the result.

Fortunately, we get a true theorem by choosing a simple rule that selects, among t_n and t_p, the time at which propagation is enabled. There are two major justifications for this:

1. Had we chosen the time of the event that occured *after* propagation was enabled, then the event might not have appeared in the observed calendar until *after* some other event that depended on it, violating safety.

2. By having propagation enabled at the same times in both the nonatomic and atomic calendars, we avoid the complication of measuring propagation delays in a nonatomic behavior, by measuring them in the observed atomic behavior instead.

Assigning 1 to y_n *enables* the n-transistors in the following stage, while assigning 0 to the same variable *disables* those n-transistors. Thus for each nonatomic assignment S (of the form $y := v$), the **enabling sub-assignment** S_E and **disabling sub-assignment** S_D depend on v, as shown in the following table:

S	S_E	S_D
$y\uparrow$	$y_n\uparrow$	$y_p\uparrow$
$y\downarrow$	$y_p\downarrow$	$y_n\downarrow$

If we define t_E as the timestamp of S_E, and t_D as the timestamp of S_D, we can write our observation rule ϕ as a projection:

$$\phi(r, t_E, t_D) \stackrel{\text{def}}{=} (r, t_E)$$

6.6 Transmission Errors Are Ruled Out

If the observation theorem holds, the sub-assignments to a variable implementing one assignment cannot be interleaved with the sub-assignments to the same variable implementing another assignment. For example, if both $x_n\uparrow$ and $x_p\uparrow$ are 1, they must both be assigned 0 before either can be assigned 1 again. A **transmission error** is a violation of this assignment pattern. It has been proven that if transmission errors are the only type of errors possible, then there are no transmission errors[5]. Since we make no such assumption in proving our observation theorem, we will need a stronger result about transmission errors.

In the language of P/N-fork calendars, we will now show that for each node y, the set of all time intervals of the form $[t_D, t_E]$ and $[t_E, t_D]$, taken over all events $(\cdots y := \cdots, t_E, t_D)$, do not overlap. We define a **transmission error** as an event whose time interval overlaps with the time interval of an earlier event assigning to the same node. We prove the following fact about transmission errors:

Theorem 3 (Transmission Errors) *If the observation theorem can be proved assuming no transmission errors, then it holds even if transmission errors are modelled.*

Assume the observation theorem is proven using the assumption of no transmission errors. Suppose (for a contradiction) that some execution L has transmission errors, and consider the earliest sub-event e of the earliest transmission error. We can assume that $e = S_E$ (if $e = S_D$, then e was enabled at the same time as S_E, so there is a similar execution where S_E uses the timestamp of S_D). S_E is a transmission error because it comes between two sub-events T_D and T_E that assign the same target as S_E but the opposite value. By the nonatomic stability portion of the observation theorem, the guard of T holds continuously between the timestamps of T_D and T_E. Clearly the guard of S holds at some point in this interval, by atomic stability. Therefore there is atomic interference. □

We will assume the absence of transmission errors when we prove the signal lemma (sections 6.11 and 8.1).

6.7 Method of Ruling Out Execution Errors

The second clause of the observation theorem states that given any execution L of the implementation, its observation H (a calendar in the specification model) must be an execution. In other words, it must satisfy safety and progress.

Progress of H is guaranteed by the following:

1. **Signal Lemma**. Find a simple relationship between state in L and state at the same time in H. The signal lemma holds until the time of the first transmission error, so transmission errors must first be ruled out.

2. Check that **indefinite enabledness** (i.e., enabledness forever after some time) in H implies indefinite enabledness in L.

Safety of H is proved by contradiction: Suppose H contains an unsafe event, and consider the earliest such event. This event was enabled in L but not in H. Before we take this to a contradiction, we simplify the problem by partitioning it into two V_{DD} regimes.

6.8 The V_{DD} Parameter

As discussed in section 5.1, we analyze an analog signal using the times when the signal crosses the V_{Tn} and V_{Dp} thresholds. If we change V_{DD}, we change the relative positioning of these two thresholds, and hence we change the time ordering of the two sub-assignments:

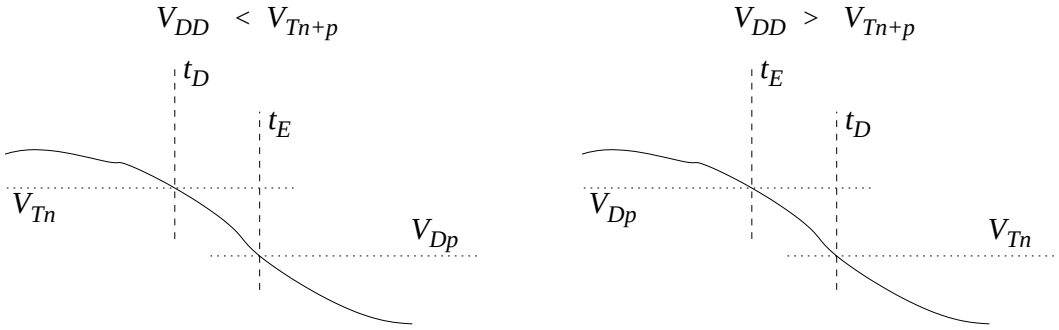


Figure 15: *The relative timing of enabling and disabling sub-assignments is affected by the choice of* V_{DD}.

The change from one behavior to the other occurs when $V_{Tn} = V_{Dp}$, in other words, when V_{DD} equals the threshold sum V_{Tn+p}, defined thus:

$$V_{Tn+p} \stackrel{\text{def}}{=} |V_{Tn}| + |V_{Tp}| \tag{1}$$

6.9 Low V_{DD} Versus High V_{DD}

Notice that by talking about the times of S_E and S_D (instead of directly about the times of transitions on x_p and x_n), we have obtained an S_E, S_D ordering which (for fixed V_{DD}) is independent of the transition sense:

case	V_{DD} range	S_D, S_E order
Low V_{DD}	$V_{DD} < V_{Tn+p}$	$S_D; S_E$
High V_{DD}	$V_{DD} > V_{Tn+p}$	$S_E; S_D$

The proof of the observation theorem will differ substantially between the low V_{DD} and high V_{DD} cases. Intuitively, this difference comes from the effect that the S_E, S_D ordering has on the semantics of the state between these two statements. If S_E comes first, the intermediate state (which we call "X") of the nonatomic implementation of the target variable enables (i.e., turns on) both N and P transistors, while if S_D comes first, the intermediate state ("Z") enables neither transistor type. For short, we will sometimes say that a "nonatomic variable x" has the value X or Z; what this means is that the variables x_n and x_p have the values in the following table:

x (nonatomic)	x_n	x_p	notes
0	0	0	stable state, all transistors see 0
X	1	0	intermediate state, high V_{DD} only
Z	0	1	intermediate state, low V_{DD} only
1	1	1	stable state, all transistors see 1

6.10 Progress Holds in the P/N-Fork Model

Before we restrict the P/N-fork model, we separately handle the portion of the observation theorem which states that atomic observations of nonatomic executions satisfy progress. Proving this portion without restricting to the low- or high-V_{DD} case will make it easier to integrate the two cases in section 10.

Theorem 4 (Progress) *If the observation theorem can be proved except for the progress portion, then the progress portion holds as well.*

Suppose not: suppose some nonatomically stable and noninterfering execution L without transmission errors has safe atomic observation H not satisfying progress. After some time t_0, some rule $g \rightarrow S$ is indefinitely enabled in L but not in H. This is because of some nodes used in g that are Z in L. Since there are no transmission errors, each node is only temporarily Z, and there is an infinite sequence of intervals during which some of these nodes have value Z, covering (t_0, ∞). Each interval has the form $[t_D, t_E)$. We produce a new execution in which the t_D are delayed to come within ϵ of the t_E. This produces pulses in g, contradicting nonatomic stability. □

6.11 Observation Theorem Proof, Low V_{DD} Case

The low-V_{DD} case is much simpler to analyze than the high-V_{DD} case, because the unsafe behavior shown in Figure 8 does not occur, as there are no X states (only 0, 1, and Z). This will allow us to prove the low V_{DD} observation theorem without using any assumption about slewrate or feedback delay timing. We will only have to assume that there are no **self-invalidating gates**. In other words, no node can be the input and output of a single gate, as guaranteed by Martin Synthesis[1].

We now prove the following four results about the nonatomic execution L of any stable, noninterfering, self-invalidating-gate–free PRS:

1. L is nonatomically stable: For each rule $g \rightarrow S_D; S_E$, whenever g holds, it continues to hold until S_E.
2. L is a noninterfering execution.
3. Safety: H (the observation of L) is safe.
4. H satisfies progress.

As discussed in subsection 6.7, we begin an observation theorem proof by relating the state of an implementation execution with its observation:

Lemma 1 (Signal, low V_{DD} case) *Consider any low V_{DD} execution L (i.e. an execution with all events satisfying $t_D < t_E$) and its observation H, and any signal x. Whenever x has the value v in H, the value in L is either v or Z.*

Proof: By induction on timestamps. Recall from Section 6.6 that there are no transmission errors. Clearly the hypothesis holds initially. After that, each assignment to x is implemented as S_D followed by S_E. The first sub-assignment makes $x =$ Z in L, preserving the hypothesis. By our observation rule, the second sub-assignment assigns the same value (0 or 1) to x in both L and H, also preserving the hypothesis. □

The low-V_{DD} signal lemma says that H always enables at least the guards enabled in L, if not more. As we will see, this property guarantees that if H is a stable, noninterfering execution, then so is L. For easier integration with the high-V_{DD} case in Section 10, however, we will get this result by way of argument about L and a more-enabled *nonatomic* execution L', rather than the atomic observation H. For the low V_{DD} case, we will simply let $L' = H$, but the following lemma does not assume that:

Lemma 2 (More-Enabled Execution) *Suppose L is a nonatomic execution with no transmission errors, and consider any nonatomic calendar L' which is obtained from L with only the state modifications $Z \to 0$ and $Z \to 1$, by delaying each S_D event to the time of the associated S_E event. L' is nonatomically safe.*

Proof: Each S_D event is enabled at time t_E in L, by Nonatomic Enabledness (Theorem 2). The event is therefore also enabled in the more-enabled calendar L'. □

Lemma 3 (More-Enabled Transient) *Consider any L and L' as in the execution lemma. Suppose furthermore that L' has observation H, an execution. If L' is nonatomically stable and noninterfering, then so is L.*

Proof: Any interference in L is also an interference in the more-enabled execution L'.

Now suppose for a contradiction that nonatomic stablity is violated at time t. I.e. suppose that in L, some g is disabled just after time t, with no intervening S_E:

$$\text{L} : \ldots \{g\}; \ldots (\text{no } S_E) \ldots ; \{\text{time } t\}\ T; \{\neg g\}$$

Thus g is disabled as a result of some statement T executing at time t in L. If $T = *_E$ (i.e., if it is some enabling sub-assignment) then the observation has an instability, but the PRS was stable, so we must have $T = *_D$. And we must have $*_D \neq S_D$, because otherwise $g \to S$ is self-invalidating. Then we can obtain a new partial execution (up to time t) by changing $*_D; \{\neg g\}$ to $*_D; *_E; \{\neg g\}$. Notice that $*_E$ now occurs before an intervening S_E, so the observation has an instability, which is a contradiction. □

To prove the low-V_{DD} observation theorem, we first notice that L' has (for each event) $t_D = t_E$, so L' is essentially its own observation. By the more-enabled execution lemma, L' is a nonatomic execution, and therefore H is an atomic execution, which is by assumption stable and noninterfering. By the more-enabled transient lemma, L is therefore stable and noninterfering. □

7 Constrained P/N Fork Model

While we have just proved the low-V_{DD} observation theorem without assuming slewtime constraints, we clearly must introduce such assumptions in order to prove the high-V_{DD} theorem: without the assumptions, the system shown in Figure 7 has unsafe transitions, and hence the theorem is false. We therefore add the necessary assumptions, and proceed to analyze these unsafe transitions, with the goal of proving the safety portion of the high-V_{DD} observation theorem.

7.1 A Closer Look at Slewrates and Feedback Delays

To better understand the conditions under which the circuit in Figure 7 fails, we analyze the C-element in the P/N fork model, as shown below:

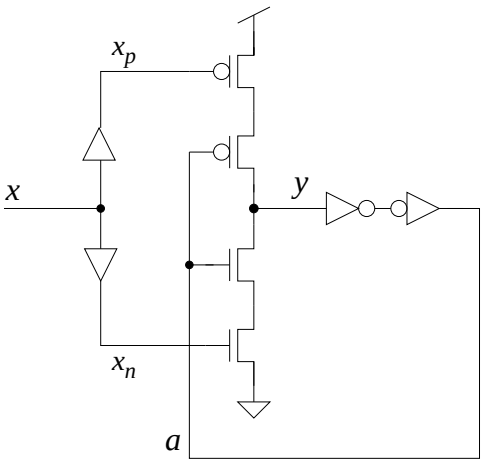


Figure 16: *System of Figure 7 in the P/N fork model*

As discussed in Section 1, there can be extra transitions on y during the time interval when x enables both N and P transistors. For example, for a downgoing transition in the P/N fork model, this interval is the time between $x_p\downarrow$ and $x_n\downarrow$. Failure can occur if the length of this interval (i.e., the slewtime of $x\downarrow$) is longer than the feedback delay from $x_p\downarrow$ back to a, as shown below:

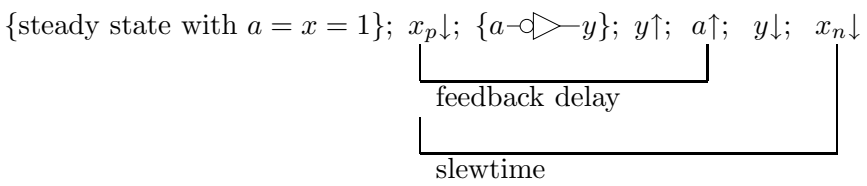


Figure 17: *Trace in which unsafe event $y\downarrow$ occurs.*

(the notation {a–▷—y} means that unless there are further x_p or x_n changes, the node y always eventually becomes $\neg a$).

In this example, the designer made a **safety assumption**. He assumed that the event following $x_p\downarrow$ by a slewtime would occur before the event following $x_p\downarrow$ by a feedback delay:

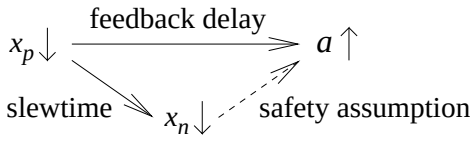


Figure 18: *Safety assumption violated in the trace shown in Figure 17*

The remainder of this paper is chiefly dedicated to proving that this assumption guarantees the correctness of the high V_{DD} implementation of *any* stable, noninterfering PRS. We will do this by supposing (for a contradiction) that some unsafe event occured despite the assumption, and we will find that in that case there was in fact no feedback path and the atomic specification allowed behaviors with instabilities.

Before we can make this argument formal, however, we must formulate the safety assumption as a property of general PRS. To be useful as a design rule, it should be a static property. And since it refers to the physical delays of a circuit's behaviors, we must **annotate** our PRS with delays, over which our static property can then be defined.

7.2 Minimum-Delay Annotations

We will define minimum-delay annotations with semantics in both the atomic timestamp model and the P/N Fork model. A priori, in both models, the absolute values of the timestamps are meaningless, as all timestamps can be scaled without changing the safety or progess of a trace. However, as noted in Section 4.2, it is convenient to give new meaning to these timestamps as physical event times. Thus our delay annotations (which represent physical delays) will directly restrict timestamps:

Definition 1 (Minimum-Delay Annotation) *A (Minimum-)***Delay–Annotated PRS** *is a PRS, having rule set $\mathcal{R}$, together with the* **delay annotation** $\alpha : \mathcal{R} \to \mathbf{R}^+$. *An atomic execution has* **valid delay timing** *if and only if the time at which a rule r is enabled and the next earliest time at which r is executed is always at least $\alpha(r)$.*

We choose to define delay-annotation semantics in the atomic model, since such a semantics naturally extends to the nonatomic model: an execution has valid delay timing in the nonatomic model if and only if its observation has valid delay timing in the atomic model. This gives the expected nonatomic semantics because we chose our observation rule (Section 6.5) so that the atomic timestamp would be equal to the time at which propagation was enabled in the nonatomic model.

7.3 Propagation Paths

We now describe feedback delays in terms of the annotations developed in the previous subsection. The delay annotation semantics we have just chosen restrict the time between the enabling of a rule's guard and the next execution of the same rule. By applying this restriction along successive stages of a propagation path, we can determine delays between events separated by a propagation path:

Definition 2 (Rule Dependence) *A rule $g \to y := v$* **depends (positively) on** *rule $g' \to y' := v'$ if there is some assignment of binary values to nodes such that changing y' to v' causes an increase in g.*

Definition 3 (Propagation Path Length) *A* **propagation path** *p is any sequence of rules, such that each rule depends on the previous rule in the sequence, if any. The* **length** *$\alpha(p)$ of the path is the sum of $\alpha(r)$ over all rules r in the sequence after (i.e., excluding) the first rule.*

These are static properties, as it is not necessary to simulate the system in order to determine whether two rules are dependent and form a path.

7.4 Minimal Perturbation

For a stable production rule set, the length of a propagation path is a lower bound on the time between events whose rules are the endpoints of the path. However, we do not bother proving this fact, since we will actually need a more powerful property: if we perturb any (atomic timestamp) execution by delaying an event e_1, this *need* not change any other event e_2 until after a time of at least the propagation path length from e_1 to e_2. Before we show this fact (in Section 7.5), we formalize perturbation.

We can perturb an execution H given any set E of events, and positive delay amount δ. An execution perturbed according to $\Delta \stackrel{\text{def}}{=} (E, \delta)$ is defined as follows:

Definition 4 (Δ-Perturbation) *A Delta-***Perturbation** *of H is any execution which agrees with H with the following exceptions only:*

1. *all events in E have been delayed by δ, and*
2. *some other events may be delayed.*

Definition 5 (Minimal Perturbation) *Given Δ, the Minimal (Delta-)***Perturbation** *H_Δ is the unique perturbed execution in which all transitions are minimally delayed. Formally, H_Δ is minimal in the sense that there is no Δ-perturbation H'_Δ and bijection $\phi : H_\Delta \to H'_\Delta$ such that ϕ preserves rules and never increases a timestamp.*

We now prove that this is a proper definition. In other words, we prove that H_Δ exists and is unique. Clearly a Δ-Perturbation exists: the entire execution can be translated by δ. There must be a unique minimal Δ-Perturbation; otherwise we would consider the earliest time at which two assumed minimal Δ-Perturbations disagree, and show by stability that events were unnecessarily delayed in one of them. □

7.5 Propagation Property

We can now formally state the property (introduced at the beginning of the preceding Section) that events need not be delayed until after the duration of propagation from the perturbed transition:

Theorem 5 (Propagation Property) *Consider any PRS with atomic execution H and perturbation Δ. If a transition e occured at time $t(e)$ in H but was delayed in H_Δ, then there must be a propagation path from some $e_\Delta \in E$ to e of length at most $t(e) - t(e_\Delta)$.*

To prove this property, we begin by considering a method of constructing H_Δ. We begin with an empty event set H_0, and we create a chain of partial executions by adding events from H one at a time, delaying them as necessary to satisfy Δ-perturbation and timing validity. To obtain H_{i+1} from H_i, consider the earliest event e in H that has not yet been mapped. Map e to a new event in H_{i+1} according to the following cases:

1. If $e \notin E$, and $H_i \cup \{e\}$ is an execution with valid timing, then simply let $H_{i+1} = H_i \cup \{e\}$.
2. If $e \in E$, then let $H_{i+1} = H_i \cup \{(r(e), t(e)+\delta)\}$. H_{i+1} is an execution because the new event was added after the last event in H_i. At this time, its state matches the state before e in H, because the same assignments have occured in the same order.
3. If $e \notin E$, but $H_i \cup \{e\}$ is not an execution or does not have valid timing, then move the event to $\alpha(r(e))$ after the latest preceding time at which it was not enabled. There is such a latest time because (as argued in case 2) e is enabled eventually.

Case 2 assumes that events are never delayed by more than δ. This can be shown by induction on i: the hypothesis holds initially, and in each of the three cases the hypothesis is preserved.

We first prove that the limiting execution $\bigcup_i H_i$ is a minimal perturbation, so that $H_\Delta = \bigcup_i H_i$. Suppose otherwise, and consider the first event e that's earlier in H'_Δ. H'_Δ is a Δ-perturbation, so $e \notin E$. e was moved in H_Δ to satisfy delay timing or safety. But by assumption, H_Δ agrees with H'_Δ up to time $t(e)$, so e violates delay timing in H'_Δ.

Now we prove the theorem, by constructing a propagation path from e_Δ to e, for any e delayed in H_Δ. Let $e_0 \stackrel{\text{def}}{=} e$. We construct a propagation path in reverse: $r(e_0)$ is the end of the path. By induction, we construct a sequence of $k+1$ *delayed* events $e_k, \ldots, e_0$, with $e_\Delta \stackrel{\text{def}}{=} e_k \in E$, and with $r(e_k), \ldots, r(e_0)$ a path. The base case holds as e_0 is delayed and forms a trivial path. For the inductive step: e_i was delayed, but since H_Δ is minimal, e_i could not be moved any earlier, owing to rule dependence on some other delayed event e_{i+1}. The inductive step fails when some $e_k \in E$. At that point, we have constructed a path of length at most $t(e_0) - t(e_k)$. □

7.6 Slewtime Annotations

In addition to delay annotations, we add slewtime annotations to our PRS. Unlike delay annotations, however, slewtime annotation semantics only make sense in a nonatomic model:

Definition 6 (Slewtime Annotation) *A* **Slewtime-Annotated PRS** *is a PRS, having rule set $\mathcal{R}$, together with the* **slewtime annotation** $\tau : \mathcal{R} \to \mathbf{R}^+$. *A high V_{DD} P/N-fork execution has* **valid slewtime timing** *if and only if each event (r, t_E, t_D) satisfies $t_D - t_E \leq \tau(r)$.*

7.7 Interference in Combinational Transitions

The unrestricted P/N fork model yields undesired interference in combinational gates: If a single node change in a guard g simultaneously increases g while decreasing the opposing guard q, there will be interference while that node is X. We therefore restrict the behavior of gates in which a node x is used in both the PUN and PDN:

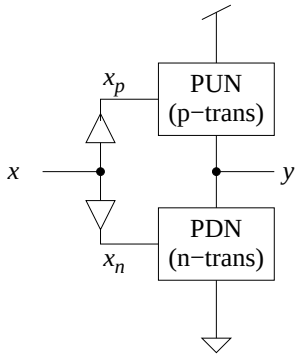


Figure 19: *In a typical gate, each input x is used in complementary guards with a common target.*

We use a previously known solution to this problem in which combinational gates have a single input threshold [2], hence effectively removing the extra copy of the input variable introduced by the P/N fork model in this case. Unlike prior efforts, however, we do not require that specific gates be identified as combinational, as some common gates exhibit both combinational and dynamic behavior, depending on the state and input changes. We instead identify each *transition* as either combinational or not:

Definition 7 (Combinational Transitions) *A P/N fork behavior L* **satisfies the combinational transition constraint** *if each gate with some guard g, some opposing guard q, and some target y satisfies the following: Consider each event with target y that executed at some time t when q was enabled. For each term that occurs both in g and in q and is X at time t, we require the transition on that node to be completed at time t.*

This definition does not in general require that transitions on combinational gate inputs be atomic, but it at least truncates their slewtime so as to remove interference on the output. The implementor is of course free to guarantee a more restrictive assumption.

8 Proof of the High VDD Observation Theorem

We begin the proof of the high-V_{DD} observation theorem by proving the progress portion of the statement. Consider any high-V_{DD} P/N-Fork execution L with valid (delay and slewtime) timing, as developed in the last section. We prove that its observation H satisfies progress.

8.1 Progress

As in the low-V_{DD} case, there is a simple relationship between the state of a nonatomic system and the state of its observation. The only difference is that during the slewtime interval the state is X instead of Z:

Lemma 4 (Signal, high-V_{DD} case) *Consider any high-V_{DD} execution L (i.e., an execution with all events satisfying $t_D < t_E$) and its observation H, and any signal x. Whenever x has the value v in H, the value in L is either v or X.*

Proof: By induction, similar to low-V_{DD} case. □

As in the low-V_{DD} case, the the only permanent states are 0 and 1, in which L and H agree; hence H progresses if L progresses. (We could have just used the Progress Theorem, but we needed the Signal Lemma anyway). □

8.2 The Slewtime Constraints

As explained in Section 7.1, the observation theorem does not hold in general for high-V_{DD} unless we assume that the slewtime of a transition is at most the feedback delay of that transition. We can now state this formally, in terms of the annotations and propagation paths we have defined:

Definition 8 (Safety Constraint) *A PRS* **satisfies the safety constraint** *if it has delay and slewtime annotations such that for all propagation paths p such that some rule r (not necessarily in the path) depends negatively on the starting point of p but positively on the final point of p,*

$$\tau(r) < \alpha(p)$$

Paths p containing (but not being themselves) cycles need not be considered, as such p are longer than the same p without the cycles, so such p add no new constraints in that case.

We will also need the following:

Definition 9 (Noninterference Constraint) *A PRS* **satisfies the noninterference constraint** *if it has delay and slewtime annotations such that for each rule r with opposing rule r', and each* negative *path p from r to r',*

$$\tau(r) < \alpha(p)$$

A **negative path** is a path in which the first dependence is negative, and the rest are all positive.

8.3 Safety

Suppose H contains an unsafe event, and consider the earliest one, e. As noted in Section 6.7, this event was enabled in L but not in H, at some earliest time t. The event was not enabled H either because it was vacuous or because the guard was not enabled. If e is vacuous in H but not in L, the difference is due to the target's being X, but this type of vacuous event was ruled out in Section 5.2. Therefore we can assume the guard is to blame.

By the signal lemma, the guard g of rule $r(e)$ can *only* be enabled if some of its input nodes are X; call this node set P. Let S_0 and S_1 denote the sets of nodes used in g having the values 0 and 1, respectively, at time t in L. Notice the following three facts:

1. For small ϵ, g becomes enabled in H at time $t - \epsilon$ if the nodes in P are changed to the values they had before their last assignment.

2. The last assignment (before time t) to each node $p \in P$, of value v, occured during the time interval $[t - \tau(p, v), t)$. Let R denote the unique set of rules responsible for these assignments. I.e.,

$$R \stackrel{\text{def}}{=} \{r : r = \cdots \rightarrow p := v, \text{ where } v \text{ is the value of } p \text{ at time } t \text{ in } H\}$$

3. g is disabled in H at time t so long as S_0, S_1 and P have their original values at time t.

We now ask the question of what happens if we delay the assignments to P (in H) such that g is enabled at time $t - \epsilon$ (by fact 1). There are two possiblities:

1. S_0 and S_1 are "protected" by feedback delays. Let S_* denote the set of latest rules affecting S_0 and S_1 at time t. If all propagation paths from R to S_* have length at least $\tau(R) \stackrel{\text{def}}{=} \max_{r \in R} \tau(r)$, then, by minimal perturbation, the changes to H do not affect S_* until after time t. With these changes, g is momentarily enabled over $[t - \epsilon, t)$, showing that the atomic specification allowed unstable behavior.

2. If some path is short enough such that some S_* is affected, then the slewtime constraint (Definition 8) is violated for some rule in S_*. □

8.4 Nonatomic Noninterference

For a contradiction, consider the earliest time t which is (arbitrarily close to) a time at which some guard g and its opposing guard q are simultaneously enabled in L. We can assume H is an execution up to time t. Since H is noninterfering up to time t, at least one of the guards, say g, is enabled only because one of its nodes is X.

As in the safety case, we attempt to delay the X nodes in g until after q. q is not affected (by time t) because of the noninterference constraint, so we find a perturbed H which is an interfering execution (up to time t), a contradiction.

But wait, don't we know that this argument breaks unless we introduce ratioed transitions? Indeed, there is a problem if the nodes whose transitions are moved in g also occur in q, and that case is ruled out with ratioed transitions. (This case did not come up in the safety argument, because the sets P, S_0 and S_1 were disjoint by construction). □

8.5 Nonatomic Stability

Finally, we must show that whenever a guard g is enabled in H, it remains true until both sub-assignments on the target y have completed, at times t_E and t_D. Assume for a contradiction there is a nonatomic instability, and consider the earliest one. We can assume H is an execution until the time of the instability, and therefore that g holds until t_E. If g did not also hold until t_D, we perturb H by delaying the target without affecting the impending guard invalidation:

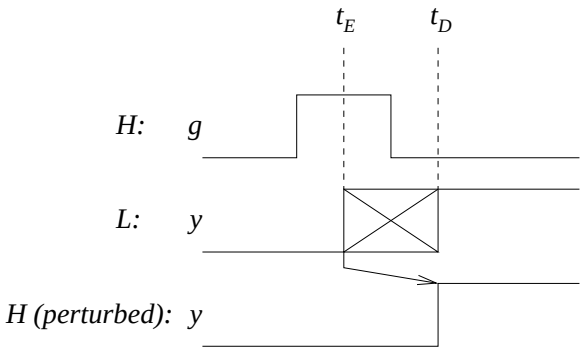


Figure 20: *Under the safety constraint, if there is a nonatomic instability, the target can be delayed without affecting the guard invalidation.*

The atomic assignment to y is then an unsafe event, which is a contradiction. □

9 Design Examples

9.1 Slewtime Constraints for CMOS Implementable Rules

So far we have not placed any restrictions on the boolean formulas used in guards. To be directly implementable as CMOS transistor networks, however, pulldown guards must be positive monotonic, and pullup guards must be negative monotonic. This restriction is called **CMOS implementability**[1]. With this restriction, a rule can only depend on a rule of opposite target value. A path with an odd number of edges connects rules of opposite target values, while a path with an even number of edges connects rules of similar target values.

Consider the high-V_{DD} safety argument (Section 8.3) for a rule in a CMOS-implementable PRS. Suppose (without loss of generality) that the unsafe rule is a pulldown rule. The rule was unsafely enabled because nodes in the set P were X; in this case their last assignment was of value 0 (because by assumption they are 0 at time t in H). To create a contradictory pulse, some nodes in S_1 (which were last assigned 1) must not be delayed. So the argument only relies on the path from P to S_1, which has an odd number of edges.

The Noninterference constraint is similar. Since all of our examples are CMOS Implementable PRS, we only consider paths of an odd number of edges in this section. For example, we would not need (or want) to consider the path from x to y in the following circuit:

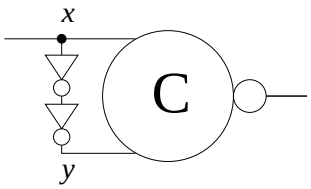


Figure 21: *Paths of an even number of edges need not be considered in CMOS-Implementable PRS*

It is important that we rule out such paths from consideration, because short paths of this form common in completion trees.

9.2 Cyclic Constraints

The most common path requiring a constraint is the path formed when an odd number of gates form a cycle. If each gate in a cycle has the property (as is usually the case) that each input is used in both the PUN and PDN, then there will always be both a safety constraint and a noninterference constraint for that cycle:

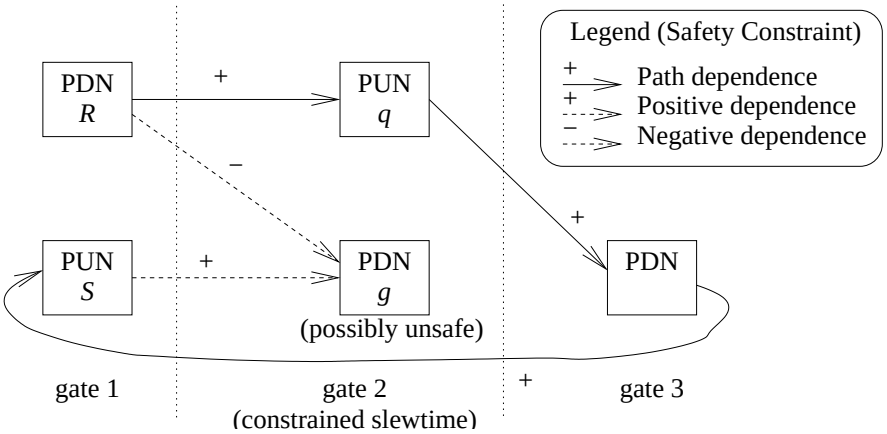


Figure 22: *Safety constraint for a gate in a ring of 3 gates.*

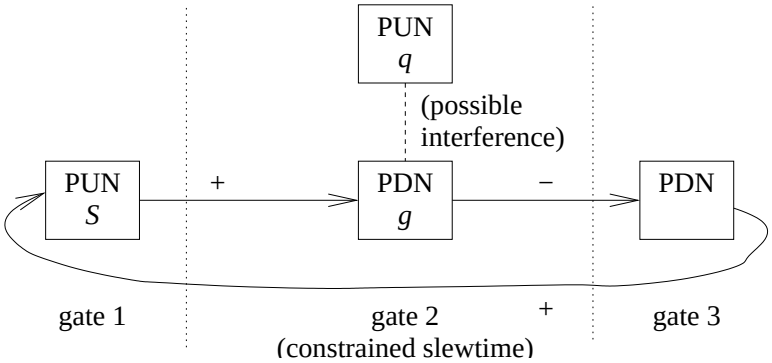


Figure 23: *Noninterference constraint for a gate in a ring of 3 gates.*

As the cycle increases in number of gates, the two constraints converge to a single constraint (and of course that constraint becomes easier and easier to satisfy).

In such cycles, the path from R to S_* is through r. In this case, the path has the same length as a cyclic path through S_*, since in our annotation system, r has the same delay from all rules that it depends on:

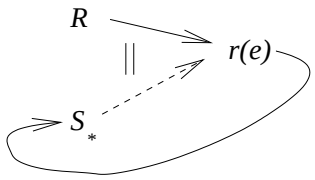


Figure 24: *The length of a path from R to S_* is the same as the length of a cycle through S_**

This might be a useful simplification for a design-rule checking tool.

9.3 Isochronic Non-Directed Cycles

The path from R to S_* is not always through r. In fact, some of the shortest (and therefore hardest to implement properly) paths connect an isochronic fork[1] to another node feeding the same gate. For example, the PRS represented below is atomically safe in the case where initially $x = 1$ and $y = 0$, and then x changes:

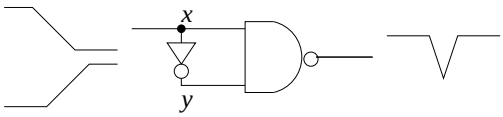


Figure 25: *Glitch caused by atomically safe isochronic non-directed cycle.*

Such constructs must be carefully designed when implemented in real circuits.

9.4 PCHB

We now consider the Pre-Charge Half-Buffer (PCHB)[6], a high-performance domino-logic pipeline stage developed by Andrew Lines and used extensively in the MiniMIPS[3]:

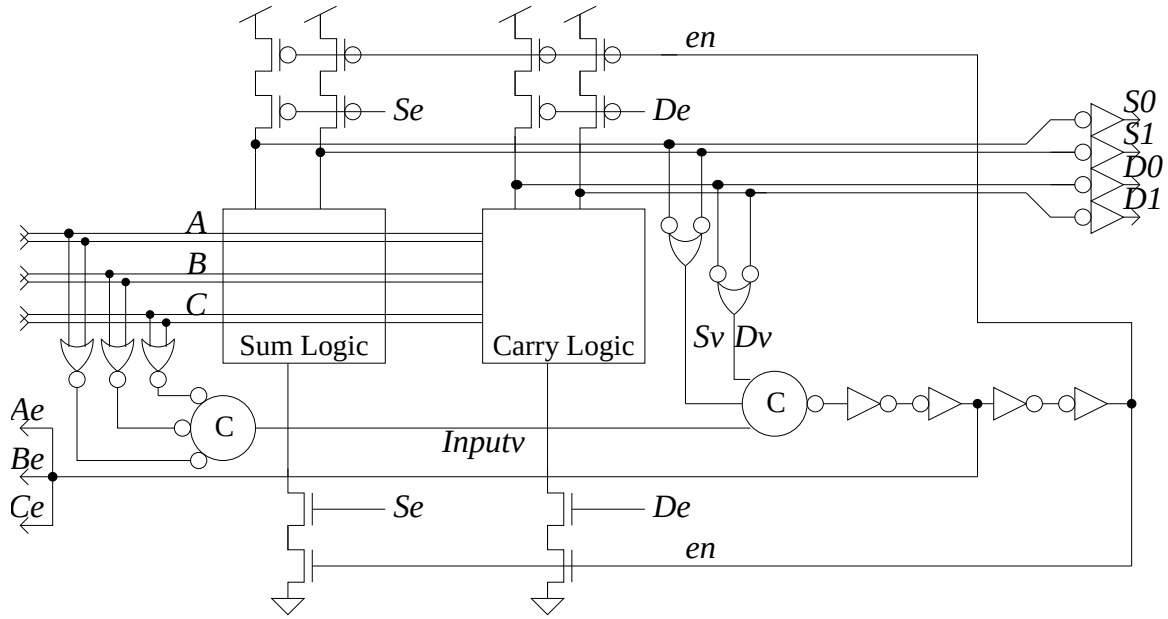


Figure 26: *PCHB Full Adder circuit.*

Counting delays in terms of gate transitions, the following 14-transition cycle has the longest delay:

$$A\uparrow; \_Av\downarrow; Inputv\uparrow; \_\_en\downarrow; \_en\uparrow; Ae\downarrow; \_A\uparrow$$

(We write only the first half of the cycle, since the second half is the same except that $\uparrow$ is swapped with $\downarrow$). This cycle occurs when the PCHB is connected to another PCHB on its left interface (A, B, C).

This cycle limits the throughput of the circuit[6]. This cycle is also a limiting constraint on slewtime (of the type discussed in section 9.2). Therefore no slewtime in the circuit can be more than 7 transitions. In the MiniMIPS, this was achieved by limiting the capacitance on each circuit node, and checking the simulation with `alint`[4].

9.5 PCFB

The Pre-Charge Full-Buffer (PCFB) implements an HSE with fewer timing constraints than the PCHB, and is used when more performance is needed or when variations in pipeline timing must be absorbed[6]:

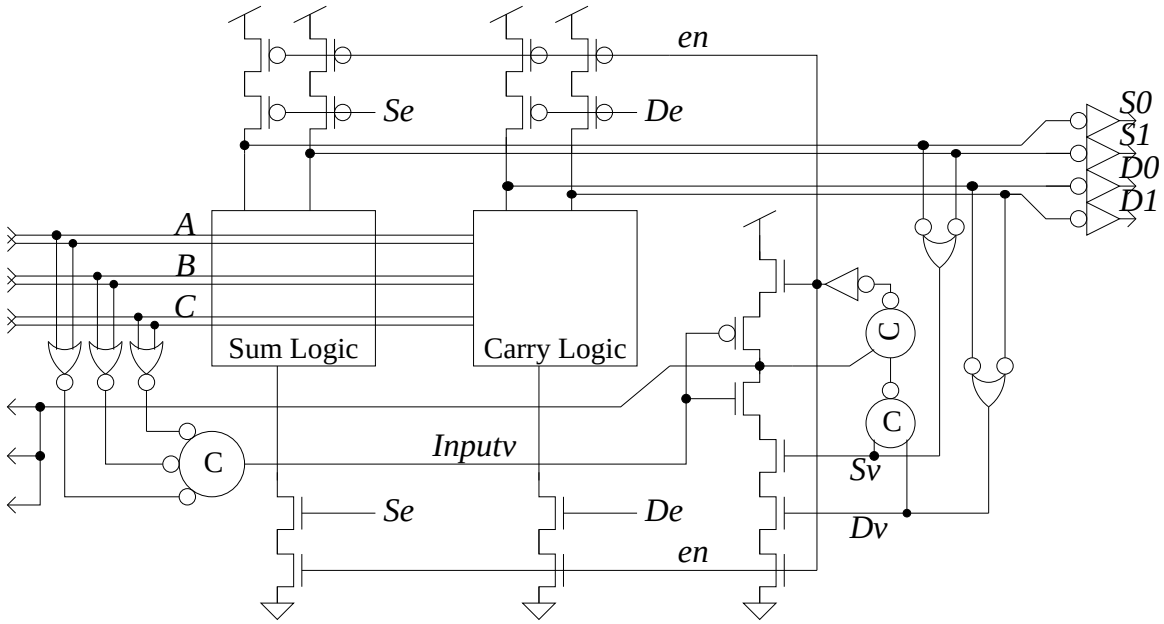


Figure 27: *PCHB Full Adder circuit.*

The **input completion** is the circuit combining the **input validity** *inputv* and **output validity** Sv, Dv to produce the input acknowledges Ae, Be, Ce. While the input completion of the PCHB awaits the validity of all inputs and outputs in both the setting and resetting phases, the input completion of the PCFB awaits the output validity only in the setting phase. This saves two transitions in the resetting phase, allowing the PCFB to run at 12 transitions per cycle.[6]

While the performance has improved by a factor of 14/12, the slewtime constraints have tightened by a factor of 7/5, as the shortest cycle through the input completion now has 5 gates on it. In addition, there is now a cycle internal to the PCFB control circuitry ($en, Ae, Ae\_$) with just 3 gates on it. This cycle does not limit performance, but imposes an even tighter slewtime constraint. This problem can be slightly eased by inserting two inverters on $Ae\_$. However, the effect of nonatomic transitions is still harder to control for the PCFB than it was for the PCHB, even considering the performance improvement.

10 Integrating the Low-V_{DD} and High-V_{DD} Theorems

We have now proven the observation theorem assuming no X states (low V_{DD} case), and we have separately proven it assuming no Z states (high V_{DD} case). Usually, for performance reasons, a chip is operated squarely in the high V_{DD} region. However, a strength of QDI circuits is that they are robust to supply voltage changes, and function properly for low V_{DD} as well. Another strength of QDI circuits is their robustness to local parameter changes; for example, part of a chip can run slower (due to manufacturing variation, say) without affecting correctness the speed when that part is not in use. It is natural, therefore, to ask whether the observation theorem holds when some parts of a chip are behaving as in the low-V_{DD} case while others are behaving as in the high-V_{DD} case.

In fact, we need not designate some circuits as high-V_{DD} and others as low-V_{DD}; the circuits can change from one regime to the other as time progresses. However, to avoid failures, we clearly must impose delay and slewtime constraints on the circuits when they behave as high-V_{DD} circuits. We now examine how to apply our high-V_{DD} timing validity definitions to general P/N fork behavior:

1. It is difficult to separate the *delay constraints*, since these apply to loops which may have both types of behaviors on a given execution. Fortunately our original definition makes sense for both low-V_{DD} and high-V_{DD}, since it only refers to the atomic observation H. We therefore assume, for our new observation theorem, that H has valid delay timing (as defined in Section 7.2).

2. The *slewtime constraints* should apply to those transitions in which S_E precedes S_D (as in high-V_{DD}) but not to those in which S_D comes first. Fortunately, if we reinterpret our original definition (from Section 7.6) as applying to general P/N fork behavior, we find that the latter type of transition is unconstrained, as desired (because $t_D - t_E < 0$ in that case).

We now prove the general observation theorem for any stable PRS with no self-invalidating rules, with annotations satisfying the safety and noninterference constraint (Section 8.2). We consider any nonatomic execution L with ratioed transitions (Section 7.7) and valid timing, according to the definitions we have just justified.

We painstakingly structured the low-V_{DD} proof into two lemmas (see Section 6.11) so that we could wrap this proof around the high-V_{DD} theorem:

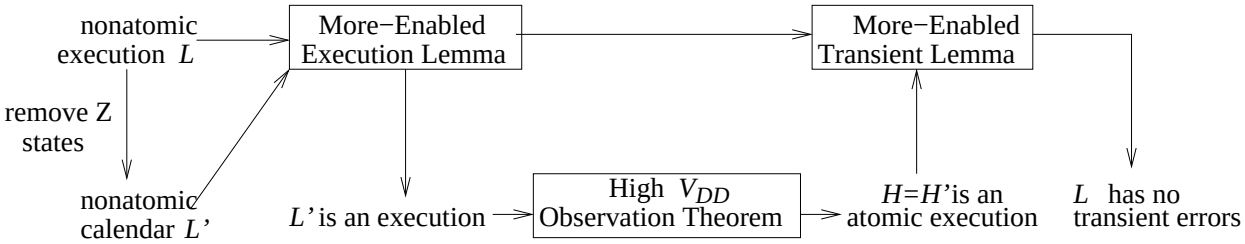


Figure 28: *Flow of structures through integrated observation theorem proof. Low-V_{DD} proof has been wrapped around the high-V_{DD} theorem, yielding a more general result.*

To apply these two lemmas, we first remove Z states from L by delaying all t_D satisfying $t_D - t_E < 0$ (for that event) so that $t_D = t_E$, obtaining L'. Clearly L' has valid slewtime timing assuming L did. By the first of the low-V_{DD} lemmas, L' is also an execution. We can therefore apply the high-V_{DD} theorem to L', yielding the first result: that H is an execution. By the second of the low-V_{DD} lemmas, L is nonatomically stable and noninterfering. □

10.1 Observation Theorem for the Multi-Atomic Models

11 Properties of Global Time

As noted in section 3.8, global time is convenient for the type of timing analysis that is necessary when non-atomic transitions are allowed. Global time allows us to easily compute the delay between any two events α and β: it's simply $t(\beta) - t(\alpha)$. Have we lost any generality of our results by employing global time? Will our theory be outdone in the same way that special relativity was outdone by general relativity? Might some behaviors have a non-embeddable curved geometry?

We prove that these problems do not arise in the setting of semantics of concurrent systems. Furthermore, we argue that semantic models without global time are unnecessarily complicated. Of course, when analysing a particular system's behavior, one is free to ignore global time, as one does in Martin Synthesis. But we encourage the use of global time in concurrent systems *semantics*, as a general rule.

Specifically, we show that global time is fully compatible with any semantics of PRS which allows non-trivial HSE specifications to be implemented. We begin with only the basic (and previously formulated[1]) assumptions about PRS semantics that we have stated in section 2.4. We briefly restate those assumptions here:

1. An execution is a set of PR firings. The firings of each rule form a sequence.
2. When a rule executes it assigns its target value to the target; node values can only change through rule execution.
3. An execution must satisfy liveness and safety.

11.1 The Successor Relation

To construct an execution (or to decide if a calendar is an execution) we must repeatedly evaluate PRs. A PR is always evaluated in some context. We do not have to assume that this context is global. For simplicity, we will assume that the context involves all terms appearing in the PR's guard. We argue, using only this assumption, that PR firings should be related by a successor relation (such as the one previously used[1]).

Suppose a rule $r = a \rightarrow \cdots$ executes. We assume r must be evaluated in a context where a is well-defined. Clearly, this implies that it is executed after some rule $\cdots \rightarrow a\uparrow$ but before the following $\cdots \rightarrow a\downarrow$. Continuing the example, suppose the aforementioned firings are named r_6, $a\uparrow_3$, and $a\downarrow_3$. We can express the ordering constraint on these firings as follows:

$$a\uparrow_3 < r < a\downarrow_3 \tag{2}$$

11.2 The Partial Order

We now claim that the "<" relation should be transitive. Suppose for a moment that the relation were not transitive, and consider the following PRS:

$$\begin{array}{rcl} a & \rightarrow & b\downarrow \\ \neg b & \rightarrow & a\uparrow \end{array} \tag{3}$$

Suppose the initial values are $a = 0$ and $b = 1$. Since no rule is enabled, we expect this to be a steady state. However, the following set of relations are an execution:

$$\begin{array}{rcl} a\uparrow_0 & < & b\downarrow_0 \\ b\downarrow_0 & < & a\uparrow_0 \end{array} \tag{4}$$

And we find the contradiction that $a\uparrow < a\uparrow$. Just about the simplest way to avoid this problem is by requiring the "<" relation to be transistive[2]. We can define the partial order "$\leq$" which holds when either "<" or equality holds.[2].

We conclude that the partial order is one of the weakest possible definitions of an execution. There are weaker definitions (section 11.7) but these are considerably more complicated, and yet they will not escape our general results about global time.

11.3 Multiple Partial Orders May Be Required

The behavior of a stable, non-interfering PRS is not in general expressible as just a single partial order. Consider the following PRS:

$$\begin{array}{rcl} \textbf{true} & \rightarrow & a\uparrow \\ \textbf{true} & \rightarrow & b\uparrow \\ a \vee b & \rightarrow & y\downarrow \end{array} \tag{5}$$

No partial order can contain the following two sequential executions:

$$\begin{array}{llll} a\uparrow; & y\downarrow; & b\uparrow \\ b\uparrow; & y\downarrow; & a\uparrow \end{array} \tag{6}$$

without also allowing unsafe calendars. In particular, to contain both of the above executions, the partial order can contain neither $a\uparrow \leq y\downarrow$ nor $b\uparrow \leq y\downarrow$. Partial order semantics of a PRS must therefore consist of a *set of possible partial orders.*

11.4 Topological Embeddability in Global Time

We now show that a set of partial order behaviors is equivalent to a set of embeddings on the real number line. Thus we do not lose any generality (compared to a partial order model) by using the timestamp model we introduced in section 4.2.

Furthermore, we prove that there is a one-to-one correspondence between sets of partial orders and the (simpler) nonempty sets of embeddings on the real number line having topologies given by those partial orders. This amounts to two statements:

1. Every partial order (over a countable set of events) has at least one embedding of the events on the positive real number line which is order-satisfying, and in which events a and b have the same timestamp if and only if $a \leq b$ and $b \leq a$.

2. No two partial orders have exactly the same set of embeddings.

To prove the first result, we start with an empty embedding (i.e., no event has a timestamp yet). Since the events form a countable set, we can insert them one at a time, maintaining the invariant that our embedding satisfies the hypotheses. We insert each new event e immediately after the latest event l such that $l \leq e$, except that we let $t(l) = t(e)$ if and only if $e \leq l$.

We now show that this procedure maintains the invariant. Clearly the invariant holds for events occuring after (or at the same time as) e. Now suppose some event l' occurs strictly before e. We must rule out the possibility of $e \leq l'$, so suppose (for a contradiction) that this is the case. e was inserted immediately after l, and l was inserted immediately after some other event, and this chain $l' \leq \cdots \leq l \leq e$ eventually goes back to l'. By transitivity, therefore, $l' \leq e$. We have reached the contradiction that $t(l') = t(e)$. □

To prove the second statement, we show that if we are given a partial order O not containing the relation $a \leq b$, then there is an embedding of this partial order in which $t(b) < t(a)$. This implies the result, because if two partial orders O, O' differ then we can assume O lacks such a relation contained in O'. Clearly the constructed embedding of O will not be an order-satisfying embedding of O'.

Consider the minimal partial order M which contains $O \cup (b \leq a)$. M contains only relations in O plus transitive chains including $b \leq a$. Suppose (for a contradiction) that $a \leq b$ is such a chain. Then the relation $a \leq b$ must have already been included in O, which is a contradiction. Therefore the embedding of O has $t(b) < (a)$. □

11.5 Geometrical Embeddability in Global Time

We have just proven that even if timing is of no concern and the underlying model is a set of partial orders, there is still no loss of generality in introducing global time. Now we consider the case where delays are of concern, and we prove again that there is no loss of generality in introducing global time, though time-varying delays may be required.

We make only the following local assumptions:

1. Whenever the delay along a path through the circuit must be locally evaluated (as in section 3.8), it is always a sum of locally evaluated delays along the edges of the path. Each edge of a path is an ordered pair of events (e, e') whose rules have direct positive dependence.

2. The delay depends only on the edge (e, e'), not on the path through the edge.

3. If the delay along two paths through the execution must both be locally evaluated, and both paths have the same endpoints, then both paths have the same delay.

These assumptions are unfortunately not enough to prove that a global time directly gives the delay between any two events as $t(\beta) - t(\alpha)$. For example, there might be two disjoint chains of locally evaluated paths between events (e, e'), and we do not assume that the delays along the two chains are equal. We could make this assumption and prove the result.

However, to be more general, we observe that to implement non-trivial specifications, events in an execution must be partially ordered, and we can embed such an execution in such a way that nonzero delays are mapped only to nonzero delays. Therefore, we can locally scale all delays in a given execution, so that our constructed embedding has the specified geometry. We can then perturb the scaling function and explore the space of possible scaling functions (for example, in an attempt to globally constrain the scaling factor).

We conclude that even if a concurrent system were wrapped around a black hole and had to be analysed using general relativity, we would still be able to analyse delays using global time, assuming the executions are still partial orders (this may amount to the conjecture in general relativity that there are no cycles going back in time).

11.6 Zeno's Paradox

Recall (section 4.2) that we require the timestamps of a calendar to be sortable into a sequence. In other words, infinity is the only limit allowed. This assumption ensures that the firings of a single rule form a sequence. It is also essential in order for all of our inductive proofs to work.

Consider the PRS shown in Equation 4. Suppose the initial values are $a = 0$ and $b = 1$. As before, we expect this to be a steady state. Consider the following atomic "calendar":

$$\left\{(a\uparrow, 1), \left(b\downarrow, \frac{1}{2}\right), \left(a\uparrow, \frac{1}{3}\right), \left(b\downarrow, \frac{1}{4}\right), \ldots\right\} \tag{7}$$

Each event is "explained" by the previous one. However, we never get to the bottom of the explanation: there was always some earlier event that was infinitesimally closer to 0. Just as with intransitive successor relations, these internal limits give behavior not allowed by the specification. Infinitesimal delays are also non-physical. Therefore we rule them out.

11.7 Weaker Definitions of an Execution

We have shown that a single partial order can be represented by a number of global-time–embeddings. Is it possible that a number of *partial orders* might be representable as a single construct? Indeed, this has been shown repeatedly. The **eXtended Event-Rule (XER) system**[11] is such a construct. In fact, the semantics of stable PRS allow only a single behavior for the XER system[11]. The XER system is therefore among the weakest possible definition of an execution, in terms of the number of behaviors it allows.

In addition to the XER system, there are formulations of an execution in which one can select only the information that is of concern to the system designer. For example, stable PRS semantics allow only a single behavior for any continuous, interchange-invariant property of the execution. The sequence of values communicated on a chip's output channel is an example of such a property.

To use global time with any of these formulations, simply represent each super-execution by a set of global-time–embeddings, just as was done with partial orders.

References

[1] Alain J. Martin. Compiling Communicating Processes into Delay-insensitive VLSI circuits. *Distributed Computing*, **1**(4), 1986.

[2] Alain J. Martin. The Limitations to Delay-Insensitivity in Asynchronous Circuits. Alain J. Martin. *Sixth MIT Conference on Advanced Research in VLSI, ed. W.J. Dally*, 263-278, MIT Press, 1990.

[3] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nyström, Paul Penzes, Robert Southworth, Uri Cummings, Tak Kwan Lee. *The Design of an Asynchronous MIPS R3000 Microprocessor*. Proc. 17th Conference on Advanced Research in VLSI, 164-181, IEEE Computer Society Press, 1997.

[4] Mika Nyström. `alint` (software), Department of Computer Science, California Institute of Technology, 1997.

[5] A. Degloria, P. Faraboschi, M. Olivieri. Design and Characterization of a Standard Cell Set for Delay Insensitive Design. *IEEE Transactions on Circuits and Systems - II*, Vol. 41, No.6, June 1994

[6] Andrew Matthew Lines. *Pipelined Asynchronous Circuits*, Master's Thesis, 1995. Caltech CS-TR-95-21.

[7] Alain J. Martin. Tomorrow's Digital Hardware will be Asynchronous and Verified.Invited paper, *Information Processing 92*, (Proc. IFIP Congress 1992), 684-695, Vol. 1, 1992.

[8] Karl Papadantonakis. *Stable Production Rule Sets are Deterministic*, Caltech CSTR, May 2003.

[9] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, and P.J. Hazewindus. The Design of an Asynchronous Microprocessor. *ARVLSI: Decennial Caltech Conference on VLSI, ed. C.L. Seitz, 351-373, MIT Press, 1989.*

[10] Alain J. Martin, Mika Nyström, Karl Papadantonakis, Paul I. Penzes, Piyush Prakash, Catherine G. Wong, Jonathan Chang, Kevin S. Ko, Benjamin Lee, Elaine Ou, James Pugh, Eino-Ville Talvala, James T. Tong, Ahmet Tura. The Lutonium: A Sub-Nanojoule Asynchronous 8051 Microcontroller. *Proc. 9th IEEE International Symposium on Asynchronous Systems & Circuits (ASYNC), May 2003.*

[11] Tak Kwan Lee, *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits.* Caltech PhD. Thesis, May 1995.

Index