Rigorous Analog Verification of Asynchronous Circuits

Thesis by

Karl Papadantonakis

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

January 13, 2006
(Defended December 1, 2005)

© January 13, 2006
Karl Papadantonakis
All Rights Reserved

Acknowledgments

After nearly six years of my most focused work ever, I have reached what I believe to be an important milestone in the theory of asynchronous VLSI. In my first few months at Caltech it became clear to me that this milestone – a method for rigorous verification of asynchronous circuits, sufficient for arbitrary computation – was still a dream, but clearly a possibility. I shared this dream with many great thinkers at Caltech who have known the dream long before me and made significant contributions. I merely assembled the pieces into one possible complete picture.

While I claim credit for an original formulation, analysis, and solution to the problem, the vision and a large part of the intuition came from my advisor, professor Alain J. Martin, and his Caltech Asynchronous VLSI Group. A key element of the group's research is the synergy of two previously disjoint fields of study. Alain is unique in his advanced knowledge of both fields and in his vision to combine the two. Those two fields are hardware implementation in VLSI and hardware specification as distributed systems. Alain has dedicated the last two decades to combining specification and implementation concepts, yielding the best possible VLSI design methods. Every result has been carefully considered from both the specification and implementation point of view. My work concerns the implementation of the underlying operational model and could never have succeeded without Alain's original vision that the model would withstand the critical dual test of both specification and implementation.

Beyond Alain's vision and leadership, it is the years of work by his many past and present graduate students that really put the model to the test. On my first day at Caltech, Mika Nyström – then still a graduate student – was a mentor and friend, as is still the case today. I learned that there was more to circuit design than just writing netlists; there were circuit templates and software tools and formal methods. Mika had developed all of this knowledge from his work on the Caltech MiniMIPS processor.

The MiniMIPS processor and its high-performance circuits are largely a creation of Andew Lines and Uri Cummings. From Andrew's master's thesis and design documents, I learned (with Mika's helpful discussions) how to build asynchronous logic and design a processor. I also learned from simulation languages developed by Marcel van der Goot, Rajit Manohar, Eitan Grinspoon, and Mika Nyström. This combination of design and tools knowledge is what allowed me to produce a new simulation tool and top-level design for the Lutonium, the group's next major processor design after the MiniMIPS.

Mika's efforts in the verification of the MiniMIPS led to his invention of `alint`, a program that checks several correctness properties of analog implementations of asynchronous circuits. Mika and the async group successfully employed `alint` to ensure that the MiniMIPS worked. The idea of constraining slewtimes is one of the most important parts of my theory, on that came from Mika and `alint`.

I believe that the main contribution of this thesis is the expression of analog implementation in terms of elementary mathematical concepts: functions, differential equations, sequences, and the properties that go with them. Professor John Hamal Hubbard, my undergraduate advisor at Cornell, taught me how to think effectively about such mathematical constructs. In his calculus and analysis classes, I was exposed to a unified study of both linear and non-linear systems viewed as both differential equations and iterative systems. The analog verification problem seemed like a perfect match for this type of analysis: the analog model is differential equations, and the digital model is an iterated system, and they had to be unified.

In my undergraduate research with Hamal on the parameter space of Hénon mappings, I gained experience thinking about the trajectory and parameter spaces of multi-dimensional systems. I must admit that I found the analog verification problem to be much harder than any mathematical problem I had ever tackled before in my undergraduate mathematical training. However, I believe that exposure to similar classes of problems contributed to my capability to solve the analog verification problem.

I would like to thank specific people for feedback and ideas that contributed directly to my progress. I would like to thank my committee, Alain J. Martin, André DeHon, K. Mani Chandy, Erik Winfree, and Jason Hickey, for providing constructive feedback. Alain J. Martin provided several group meetings for the discussion of my thesis work, as well as the opportunity to participate in group projects and papers. Piyush Prakash suggested several improvements to the first draft. As I mentioned earlier, Mika and Alain contributed many ideas fundamental to the topic. Alain's tough questions helped me stay focused on the core problem and its motiviation. Professor André DeHon helped me to learn to decide which ideas to persue first, and which ones to present first. André provided direct mentorship, the opportunity to develop my presentation skills by presenting architecture research at a conference, and feedback on technical reports which contributed to my thesis. I would also like to thank Jeri Chittum, Maria Lopez, and Mani Chandy for administrative support.

Finally, I would like to thank my parents, friends, and fiancé, Kimberly Marshall, for personal support. My mother, Gretchen Willging, and her colleague, Edward Zeidman, both math teachers, sparked my math and computer interests early on. My father, Kostis Papadantonakis, encouraged me to develop broad knowledge.

Abstract

This thesis shows that rigorous verification of some analog implementation of any Quasi–Delay-Insensitive (QDI) asynchronous circuit is possible. That is, we show that in an accurate analog model, any behavior will adhere to the digital computation specifications under any possible noise and environment timing. Unlike a traditional simulation, we can analyze all of the infinitely many possible analog behaviors, in a time linear in the circuit size. A problem that arises in asynchronous circuit design is that the analog implementations of digital computations do not in general exhibit all properties demanded by the digital model assumed in circuit construction. For example, the digital model is atomic, in a sense we define. By contrast, analog models are non-atomic, and, as a result, we can give examples of real circuits with operational failures. There exist other attributes of analog models which can cause failures, and no complete classification exists. Ultimately there is only one way to solve this problem: we must show that all possible analog behaviors obey the atomic model. We focus on CMOS implementations, and the associated accepted bulk-scale model. Given any canonically-generated implementation of a general computation, we can rigorously verify it. The only exception to this rule is that restoring delay elements must be inserted into some implementations (fortunately, this change has no semantic effect on QDI circuits, by definition). Our theorem guarantees that when any possible analog behavior is properly observed, we obtain a valid, atomic digital execution. Several rigorous verifications have been produced, including one for an asynchronous pipeline circuit with dual-rail data.

Contents

List of Definitions

List of Theorems

List of Equations

List of Figures

Chapter 1

Introduction

This thesis shows how to rigorously verify analog implementations of digital asynchronous cicuit specifications. A circuit is rigorously verified ("correct") only if there exists a proof that all possible behaviors of the circuit, in the lowest-level model, adhere to the specification. As we argue in the first section, that model should be an analog differential-equations model. To date, all practical physical circuit implementations, including Complementary Metal-Oxide-Semiconductor (CMOS), are analog, though circuit designers usually use a digital abstraction.

I am specifically concerned with digital asynchronous circuit specifications in **Production-Rule Set (PRS)** form[1]. PRS is the form of circuits produced by Martin Synthesis, a method which has yielded several high-performance implementations[2][3][4][5] of mainstream CPU architectures. By construction, such circuits are correct in a particular digital circuit evaluation model[1], which I call the **atomic model**.

PRS circuits have a canonical analog implementation in CMOS. The primary result of my thesis is a proof that these canonical implementations are rigorously correct, under conditions which are quickly checkable, and which can be guaranteed by inserting restoring delay elements into the circuit, if necessary. I apply the theorem to specific non-trivial circuits, including a dual-rail pipeline.

1.1 The Importance of a Rigorous Approach

Rigor means "precisely accurate"[6], which, in the context of mathamematical proofs, means that the result is **sound** (i.e., admits no counterexamples) in a precise and *accurate* model. All behaviors under consideration must be of a consistent, precise mathematical form, which is *accurate* to the object under study. Naturally this yields the question of whether the atomic model discussed above is accurate.

Unfortunately, not all physical circuit implementations provide the atomic transitions needed to satisfy the atomic circuit model. In fact, most non-trivial circuits generated by Martin Synthesis will fail if the operational model is non-atomic. Nontheless, circuits with non-atomic transitions can faithfully implement the atomic circuit model when combined with a few simple slewtime constraints[7] (a result reinforced in this thesis).

In Section 1.3 we will view atomicity as an **attribute** (i.e., property) of operational models. Atomicity is an important *attribute* of the digital model because the **non-atomicity** of (i.e., absence of atomicity from) the analog model can lead to failures that were not considered by the digital circuit designer. This finding naturally leads to the question of what other attributes there might be which might lead to failures that were not digitally predictable. For example, if circuits had no gain, then all nodes would reach a fixed point, leading to **nonoscillation**. If the signals were not bounded and/or had no contractive regions, there could be **overoscillation**. Other possibilities (discussed in Appendix A) include **leakage drift** and **unexpected slewtime**.

While one could look at each of these examples and try to "fix" each one individually, such an approach would never lead to a satisfying answer. We can never be sure of the *absence of undiscovered problems* until the circuit construction involves a mathematically rigorous proof that all possible physical behaviors implement the digital specification.

Rigor should not be confused with formality. Any proof (rigorous and/or formal) only applies *logically sound deduction steps* to a set of pre-existing theorems and axioms. As noted above, **rigor** implies (additionally) that all pre-existing theorems and axioms refer to a specific, accurate operational model. By contrast, **formality** is only a requirement on the *steps* of the argument: they must be purely syntactic. While this requirement makes formal proofs mechanically checkable, formal proofs can be difficult to apply. It is difficult to decide whether a formal proof applies to a particular machine, because a formal proof does not necessarily include an operational model that we can compare to the machine's operational model. Rather, the formal proof includes a set of axioms that must be checked against the machine's operational model. While these axioms are often easy to check by induction, this is not the case when applying the axioms to an analog model, as we argue above (and in Section 2.6 and Appendix A).

Therefore, *rigor* – specifically, having complete correctness proofs that refer directly to a specific, accurate underlying model – is as relevant to the soundness of analog verification as formality is. While formality can be beneficial when mechanical proof-checking is required (to detect human error), it is a central thesis of mathematics that any proof can be translated into an equivalent formal one. Therefore the primary goal should be rigor.

1.2 The Analog Model

To consider all physical behaviors, we must use (as our substrate model) the differential equations describing the circuit. Of all existing models, only differential-equation models are accurate enough to capture all the effects mentioned in the previous section (and any others that have not been specifically identified). Also, owing to inaccuracies in any such model, and external interference, we must consider not just behaviors satisfying the equations exactly, but also any behaviors that are within a **noise bound** η of satisfying the equation. We refer to such behaviors as **analog evolutions**[1].

Finally, we must allow the external environment arbitrary response times. Owing to environment timing and noise, there is not a unique behavior for a given circuit. In fact, there is an infinite number of possible behaviors, so we indeed must talk about *all possible behaviors*. The details of the analog model are presented in Section 2.5, and the details of the noise bound are presented in Section 3.3.

1.3 The Digital, Atomic, Global-Time Abstraction

Circuits constructed using Martin Synthesis are correct in the weakly-fair sequential selection model[1][9]. I call this model **atomic** (meaning "indivisible") because actions executed in this model are single events. By construction, the circuits are also correct in more abstract models involving various notions of **concurrent composition**[1]. In that case, correctness depends on the particular type of concurrent composition used. If the model is equivalent to (i.e., implements) the sequential atomic model, then I call the model **atomic**. Otherwise I call the model **non-atomic**. In general, the circuits are not correct in non-atomic models[7] (reintroduced in Appendix A.1). Therefore, we will strive to satisfy an abstraction consisting of an atomic model.

A model may or may not use **global time**. That distinction is orthogonal to the atomic/non-atomic distinction: under basic assumptions that rule out time-travel paradoxes, any atomic or non-atomic model can be stated in equivalent form with or without global time[7]. The system designer may well be better off avoiding arguments involving global time[1][10]. However, given that our task is to verify behaviors stated in terms

[1]**Evolution** is a standard term used in mathematical analysis to denote solutions (approximate or exact) to differential equations[8].

of ordinary differential equations that have a global time, our work will be simplified by introducing global time into the abstraction – with no loss of generality – from the start.

Therefore we will specifically use the **atomic timestamp model**. In this model, **executions** are defined to be all live, safe **calendars** of production rules. We define a **calendar** as any set of *timestamped assignments*, as this structure is simplest to map to physical implementations. (This structure yields an atomic model because assignments are the only possible action, with exactly one action per assignment.) **Liveness** is the property of progress, while **safety** says that the execution never does anything bad. The atomic timestamp model is the most general atomic model in which synthesized circuits are correct[7]. We discuss this model further in Section 2.4.4.

1.4 The Observation Theorem

The weakest possible useful notion of correctness for a circuit implementation is that the outputs of a correct computation can be viewed on an oscilloscope connected to the output pins (output wires) of the circuit every time the circuit is started properly. In other words, the analog signals on the output nodes can be observed, and the observed sequence of values matches the sequence of values predicted in the atomic model. In the canonical implementation of PRS (Section 2.5.6,[1]) there is no difference between the output nodes and all other nodes, so we can assume that all nodes can be observed.

An analog signal is not exactly a sequence of digital values or assignments, so to observe an analog signal we need a mapping from analog signals to digital assignment sequences, which I call an **observation rule** ϕ. The observation rule ϕ extracts events from waveforms. Once we have decided on an observation rule ϕ, it is easy to define analog correctness: a circuit is correct if and only if for all analog evolutions L, the digital calendar $\phi(L)$ is an execution in the atomic model. An **observation theorem** is a theorem that proves that $\phi(L)$ is always an execution[7].

The observation rule should be as simple as possible. This gives the simplest possible definition of correctness. The observation rule should not hide any computation. For example, if the chip is broken and produces no output values, the observation rule should not magically claim that all the correct values were produced. To achieve this property, our observation rules will separate into one spatially local rule for each circuit node. Thus, if the rule claims an output event on node y, it can only be because the output waveform for y actually did something; it cannot directly inspect the inputs that led to this activity.

In my previous investigation of non-atomic *digital* implementations of PRS, the observation rule was a simple projection[7]. Unfortunately, a simple projection does not work for *analog* implementations because the signals are not necessarily monotonic, owing to noise. Our observation rule will consist of a four-threshold automaton. In Section 3.5 we define the automaton, and show that four thresholds are necessary and sufficient.

The proof of the observation theorem is the subject of Chapter 4. The remainder of Chapter 1 is devoted to the explanation of concepts used in that proof and in Chapter 3, which derives a complete formulation of the theorem from these concepts.

1.5 Signal Containment

To prove that the observation of any analog behavior is an execution, we must bound all analog signals. For example, suppose that the digital assignment $x{:=}0$ has just completed. To prove that a subsequent $x{:=}1$ has not begun and cannot propagate, we must prove – over the time interval between the assignments – that the signal remains in a logical "0" region. This is clearly true for any observation rule in which the rule for initiation of an assignment can be expressed in terms of a signal leaving a region (see Section 3.5).

Analog circuit verification requires the definition of logical "0" and "1" regions[11]. The difficulty with self-timed asynchronous circuits is that in the absence of a regular clock, the intervals over which these regions apply depend on signal timing. Furthermore, to rule out certain failures (Appendix A) we must bound signals at all times: we care what a signal does while it is transitioning, not just where it is between transitions. The simplest way to express such a for-all-time bound is as a pair of functions $l(t)$ and $u(t)$.

To prove the observation theorem, we prove that for each node x we can construct bounds $l_x(t)$ and $u_x(t)$ from $\phi(L)$, such that the voltage signal $x(t)$ satisfies the **signal containment** conditions

$$l_x(t) < x(t) < u_x(t) \text{ for all time, } \quad \text{and} \tag{1.1}$$

$$\text{assignments to } x \text{ in } \phi(L) \text{ are safe.} \tag{1.2}$$

Recall that to prove the theorem we must show that $\phi(L)$ is both live and safe. If Equation 1.2 holds for all nodes, then we have the safety portion of the result. If Equation 1.1 holds for all nodes, then the progress portion easily follows from our *logic-gate specification* (Section 4.4.3).

1.6 Fences

The $x(t)$ are given by differential equations, which we will formally introduce in Section 2.5. Therefore we are confronted with the problem of bounding solutions to a differential equation. For 1-D Ordinary Differential Equations (ODEs)[2] , this can be accomplished using **fences**, defined[8] as follows:

Definition 1 (Lower Fence) *For the differential equation $x' = f(t, x)$, we call a differentiable function $l(t)$ a (strong)* **lower fence** *if $l'(t) < f(t, l(t))$ for all t.*

Lower fences are useful because of the following theorem[8]:

Theorem 1 (Lower Fence) *For a differential equation $x' = f(t, x)$ with solution $x(t)$ and lower fence $l(t)$ such that $l(t_0) < x(t_0)$, we have $l(t) < x(t)$ for all $t \geq t_0$.*

Similarly, an **upper fence** satisfies $l'(t) > f(t, l(t))$ and there is a corresponding theorem that an upper fence is an upper bound. See Appendix C for elementary examples.

1.7 Spatial Induction

In an acyclic circuit the output signal of any gate is determined by a 1-D differential equation in terms of the inputs to that gate. Therefore, by induction on the circuit tree, we can assume that each node x satisfies the **node hypothesis** consisting of fences l_x and u_x which satisfy equations (1.1)-(1.2). This type of induction on a circuit is used routinely in the verification of standard combinational logic[11], as we discuss in Section 2.6.2.

Unfortunately, circuits that do more than a constant amount of work must have cycles, and therefore ordinary induction cannot be applied to the multi-dimensional differential equations describing them. To solve this problem I present a new form of induction called **spatial induction**, which allows the verifier to assume that the inputs to a gate satisfy their respective node hypotheses. See Section 4.4.

[2]An ODE is a DE in which all derivatives are taken with respect to one variable, t (time).

1.8 FenceCalc™

A circuit is verified by spatial induction when every node in the circuit is assigned a node hypothesis and these hypotheses are consistent. Consistency means that for each gate, Equations (1.1)-(1.2) are satisfied for its output signals, assuming they are satisfied for its input signals.

I have developed the Modula-3 program FenceCalc™, which finds consistent hypotheses if they exist and checks their consistency. FenceCalc™ uses piecewise linear fences that can approximate arbitrary continuous functions as accurately as desired.

A circuit is typically used infinitely often, so a fence is infinite. However, FenceCalc™(being an ordinary computer program) clearly must express the fence as a finite description. This is possible because a properly executing signal alternates between two types of phases: an active **transition** (or **transient**) phase of bounded length and an **idle** (or **DC**) phase of potentially unbounded length. This allows us to express our fences as a collage of two types of **partial fences**, one for each phase:

1. **DC fences**, bounding the DC phases.

2. **Transient fences**, bounding the transient phases.

As we will see in Section 4.4.6, the node hypothesis discussed above can be expressed as a containment condition that is local to these partial fences.

The transient fences produced by FenceCalc™must be sufficient approximations to continuous fences, and it appears (both experimentally and theoretically) to be important that the element-equations for the transistors be available as continuous functions. This is in contrast to prior techniques for combinational logic where the circuit elements are characterized as the union of a handful of rectangles in the I-V plane[12]. We discuss the arithmetic aspects of this in Appendix D.3.

FenceCalc™ computes exactly one of each type of partial fence for each assignment type: It computes two transient partial-fences, for x:=0 and x:= 1, and two DC partial-fences, for x=0 and x=1.

FenceCalc™ is accessed using the interactive command language `KAVL`.[3] `KAVL` provides commands for specifying the thresholds defining ϕ, computing the DC and transient fences, and for checking the consistency of the results. `KAVL` is further discussed in Section 6.2

[3]`KAVL` (Karl's Asynchronous Verification Language) is pronounced "cavil".

Conceptually, the results of FenceCalc™are plugged into the proof of the Observation Theorem that I present in Section 4. In practice we presently are not using an elementary proof-checking system, so the user is done when FenceCalc™ terminates successfully.

1.9 Results for Real Circuits

I have successfully verified two circuits using FenceCalc™. The first circuit, shown below, consists of two ring-oscillators whose oscillations are synchronized by a Muller C-element:

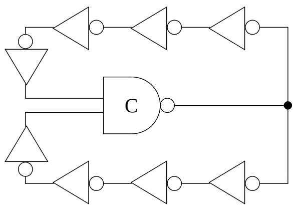


Figure 1.1: Synchronized pair of inverter rings.

The second circuit, shown below, implements a queue of bits implemented using D.E. Muller's Weak Condition Half Buffer (WCHB) Buffers[13]:

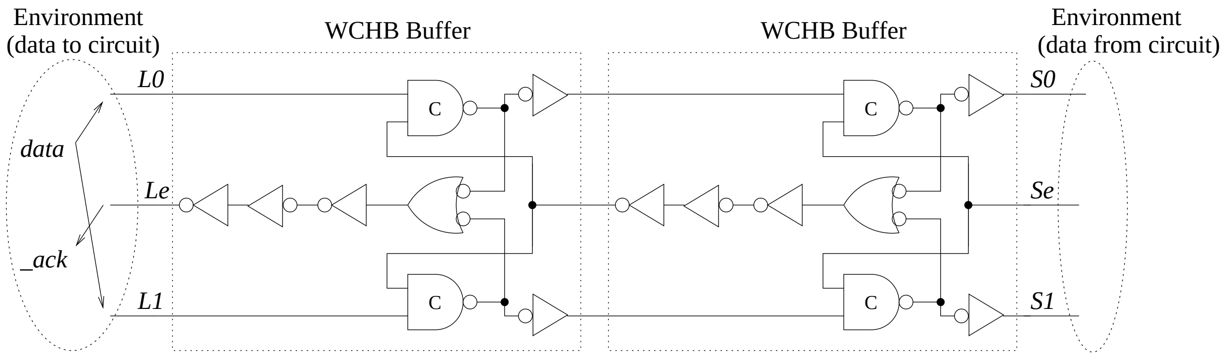


Figure 1.2: Chain of WCHB buffers.

For some of the environment handshakes, FenceCalc™ demands a minimum delay through the environment in order to verify this circuit.

For both circuits it is assumed that the transconductance of NMOS transistors is twice that of PMOS transistors, that staticizers have 5% of the strength of other gates, and that the relative sizing of different gates is in proportion to their load capacitance (see Section 2.5). I have rigorously verified these circuits assuming that a current-noise of any form whose peaks are as large as 0.1% of the maximum drive current of any gate can appear on that gate's output at any time. This is after taking error in the calculation itself into account.

The results are general. Any canonically-implemented (Section 2.5.6) PRS with a maximum fanin of two[4] can be verified using FenceCalc™ provided feedback loops are sufficiently long. In Section 7.6 we show that we can verify an arbitrary circuit if we add restoring delays to some loops. When delays are added, some of the existing calculations can be reused using incremental timing analysis[14].

In some cases, a rigorous verification exists (without adding the restoring delay elements) and yet it is difficult to find a verification, as we show in Section 7.3.2. This should not be surprising, as the mathematical set of parameters for which a circuit works is likely to have a complicated fractal boundary[8].

Finally, I have listed (in Section 3.2.4) all assumptions that I make about the element-equations of CMOS transistors so that my method can be applied to any other technology satisfying these assumptions.

1.10 Related Work

1.10.1 Speed-Independent (SI) Digital Models

A **Speed-Insensitive (SI)** circuit evaluation model has unbounded gate delays[15]. Such models can be used at all levels of digital logic design[1], from abstract high-level design[16][17][10][18] down to circuit netlists[15][1]. When the designer uses such models, analog verification is much easier than with bounded-delay models because the digital design's correctness is not affected by analog delay parameters.

Unfortunately, speed-insensitive circuits are not the most general possible class of circuits, as they do not include wire delays. **Delay-Insensitive (DI)** circuits include unbounded wire delays. DI circuits are the most general, but provably perform only trivial computations[19]. This problem can be solved by allowing only bounded wire delay[20].

As with bounded gate delay, bounded wire delay has the serious disadvantage that the digital circuit designer must deal with timing parameters. Therefore, we should ideally use the **Quasi–Delay-Insensitive (QDI)** model, which is everywhere delay-insensitive except at a (relatively) small number of nodes called **isochronic forks**, which have no delay. QDI circuits are Turing-complete[21], and Martin Synthesis produces efficient QDI circuits for an arbitrary computation[1][22].

[4]This is a limitation of the present implementation of FenceCalc™, not a limitation of the theory.

A DI or QDI circuit can be viewed as an SI circuit in which the non-isochronic forks have explicit delay elements. In this thesis all netlists and circuit diagrams are explicitly expanded so that we can analyze them using an SI model. This thesis shows that isochronic forks do not require special treatment for complete, rigorous verification. However, isochronic forks bring out the most difficult verification challenges, and previous efforts have focused on them[23].

1.10.2 Digital Verification of SI Circuits

A circuit is itself considered SI if it is correct under any timing permitted in a digital SI model. Speed-independence of a circuit is achieved directly by the correctness of automatic compilation[1][24][25][26], by automatic verification[27][28][29], by hierarchical verification [30], or by analytic verification[31] such as determinism properties[32][33].

1.10.3 SPICE Simulation

The most accurate complete behavioral descriptions of VLSI circuits are analog differential equations[34]. To obtain a complete simulation, therefore, these differential equations can be solved numerically. The SPICE simulator[35] does this quickly.

SPICE comes with standard models for all circuit elements. These models have many levels, providing incremental accuracy (and complexity). Level 0 has simple element-equations with no internal state variables except in explicit capacitors and inductors. Level 50 includes several internal state variables per element. Recent work indicates that for a small loss of accuracy, these state variables can sometimes be replaced by fewer variables by partitioning using voronoi diagrams[36].

Unfortunately, a SPICE simulation checks only a single analog behavior, and has all of the following limitations:

1. Environment timing and all noise sources must be exactly known, or infinitely many simulations are required.

2. Device parameters must be exactly known, or infinitely many simulations are required.

3. SPICE has no way to account for its own numerical error.

This last limitation makes it difficult to attach a mathematical meaning to a SPICE simulation. A simulation is not reproducible unless identical arithmetic and rounding methods are used. One cannot even conclude from a working simulation that any solution to a particular differential equation obeys the specification. In contrast to other types of calculations, it is not possible to bound the error in a SPICE simulation by simple interval analysis, as SPICE is given no specification other than the analog circuit itself, and therefore the error grows exponentially over time[8].

My solution to all of these problems is to view the noise bound (introduced in section 1.2) as a **noise budget**, which can be distributed over the above three categories, to handle all types of noise in a *finite* computation. A comprehensive noise budget that solves all three of the above problems is defined in Section 8.2.

1.10.4 Non-Atomic and Multithreshold Modeling

As we discuss further in Section 2.4, circuit evaluation models can be **non-atomic**, meaning that several events are required to properly model a single assignment action. This is possible if we begin with an abstract semantics for PRS based on a broad notion of **concurrent composition**, which can be either atomic or non-atomic[1][37]. We discuss these abstract semantics further in Section 2.2.1.

Non-atomic semantics provide an abstraction useful for analog verification when they define a **transition time**, which represents the nonzero time that it takes an analog signal to cross multiple thresholds[38][23][39]. In such models, the circuit is provably correct when the transition times are less than feedback delays[7].

The `alint` program[39], developed for the verification of the MiniMIPS processor[2], uses an advanced SPICE to check several correctness properties of analog implementations of QDI circuits. Such properties include slewtime constraints derived from multi-threshold modelling, an important idea used throughout this thesis. `alint` was successfully employed to ensure that the MiniMIPS worked without post-fabrication modification.

1.10.5 Dynamical-Space Modeling

To verify the analog safety of a circuit, we must show that all signals are bounded[40]. As discussed in Section 1.6, there is a general mathematical theory[8] which accomplishes this for non-linear differential equations (such as those describing circuits). However the theory does not directly scale to higher dimensions (as we need for circuits).

Synchronous circuits have been verified using simple piecewise-rectangular subspaces of the full multi-dimensional dynamical space[12]. Simple asynchronous circuits (oscillators) have been verified using low-dimensional polygonal projections[41]. However, unlike prior work, my algorithms run in time linear in the circuit size and work on circuits that have data rails.

Asynchronous pulse circuits can, in theory, be verified by maintaining fixed pulse envelopes that are translated in time to contain the actual pulses. The envelopes are maintained using voltage-amplification and delay properties of the circuit gates[42]. My approach to verification of QDI circuits combines these ideas with new forms of induction and stability, yielding a complete rigorous method that succeeds on several example circuits.

1.10.6 Timing Closure Methods

Prior work in analog verification uses both upper and lower bounds on gate delays. Analog correctness-only verification of QDI circuits in `alint`[39] requires only lower bounds, though upper bounds are useful for performance characterization (which is optional for QDI). Meeting these bounds is called **timing closure**. The delays themselves are a function of the relative strength of a transistor to the load it drives, known as **logical effort**[43]; we use a simplified version of this quantity that we call **relative transconductance**.

The effects of individual gate delays on the overall behavior of a QDI system have been analyzed in various digital models, based on computation graphs[44][45].

1.10.7 Symbolic Learning Algorithms

Our verification methods are based on the formulation of hypotheses that ultimately satisfy a set of necessary and sufficient conditions. As we compute these hypotheses, we **symbolically** represent our current knowledge about the hypotheses at every step of the computation. Such an approach, used in the context of analog verification of synchronous circuits, has been viewed as an application of **learning algorithms**[46].

1.10.8 Multi-Ring Systems

An asynchronous system can be viewed as a multi-ring system[47][48]. Several types of analog failures are revealed on a single ring[41]. For example, if a circuit fails by stopping all activity, that failure will be detected as a ring which fails to oscillate. Similarly, spurious activity can sometimes be viewed as a ring which oscillates too much. However, in Section A.5 we show that not all analog failures reveal themselves on rings. Therefore multi-ring abstractions are of limited use when carried into the analog domain.

1.10.9 Fault Testing and Correction

There exist methods that produce circuit designs that find and correct bit-flip errors that occur as a result of hard errors[49] (from manufacturing flaws) and soft errors[50] (from acute noise). I presently do not incorporate these methods into my theory because they are not in widespread use. However, there does not appear to be a fundamental reason why they cannot be included in a future version of the theory.

Chapter 2

Background

As noted in the introduction, we are assuming that our circuits have been generated through Martin Synthesis. This guarantees delay-insensitivity properties that we discuss in Section 2.1. It also means that our circuits are given in PRS form, which we discuss in Section 2.2. In Section 2.3 we walk through a specific example of Martin Synthesis which yields a PRS circuit for a **buffer** (i.e., a pipeline stage). In Section 2.4 we illustrate the importance of the **atomicity** of the operational model for PRS by giving a circuit that fails in a non-atomic model.

In Section 2.5 we discuss CMOS, the predominant implementation technology. We discuss how CMOS is accurately modeled using differential equations and how PRS is **canonically compiled** into CMOS. Finally, in Section 2.6 we discuss traditional verification of synchronous CMOS circuits and compare it to our asynchronous verification.

2.1 Quasi–Delay-Insensitivity (QDI) and 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 started with. The design is initially specified as a sequential program and transformed through a sequence of description levels (CHP, HSE, and PRS – see below). Each description in the sequence provably implements its specification, the previous description. The final design – a VLSI chip – therefore implements the original specification.[1]

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.[19]

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.

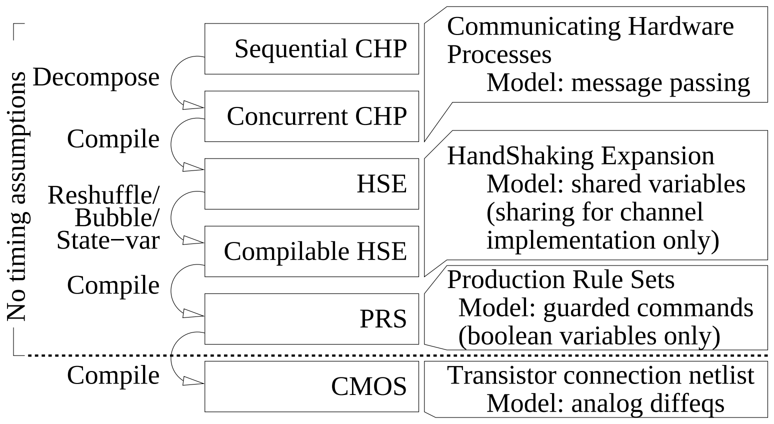

No timing assumptions
Decompose
Compile
Reshuffle/ Bubble/ State−var
Compile
Compile
Sequential CHP
Concurrent CHP
HSE
Compilable HSE
PRS
CMOS
Communicating Hardware Processes
Model: message passing
HandShaking Expansion
Model: shared variables (sharing for channel implementation only)
Production Rule Sets
Model: guarded commands (boolean variables only)
Transistor connection netlist
Model: analog diffeqs

Figure 2.1: Martin Synthesis: no timing assumptions.

Unfortunately, timing assumptions must be added in the final step of compiling PRS into CMOS circuits[7] (Section 2.4). Nonetheless, we will use the standard atomic model for PRS (Section 2.4.4) so as not to require timing assumptions to be added to any of the higher-level compilation steps.

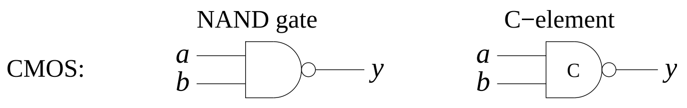
2.2 Production Rule Sets (PRS)

A PRS is a set of nodes with initial binary values[1] and a set of logic gates connecting to those nodes. We will use the simplest version of PRS in which each node y is the output of exactly one gate, which is expressed as exactly two production rules (PRs):

$$\begin{aligned} g &\;\rightarrow\; y := \textbf{false} \text{ and} \\ q &\;\rightarrow\; y := \textbf{true}, \end{aligned} \tag{2.1}$$

where g and q are boolean functions called **guards**, of other nodes in the circuit. The guards describe *when* the assignments to y occur. For a general PR of the form $g \rightarrow y := v$, we sometimes refer to the boolean v as the **target value** and the output node y as the **target node**.

As shown in Figure 2.2, 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 of the two guards is satisfied, and so the output is always (eventually) a function of the current input. I.e., the gate is combinational. Otherwise there is an input combination for which neither guard is satisfied, and it is assumed that in this case the gate holds state:



PRS:

	NAND gate	C−element
	$a \wedge b \rightarrow y\downarrow$	$a \wedge b \rightarrow y\downarrow$
	$\neg a \vee \neg b \rightarrow y\uparrow$	$\neg a \wedge \neg b \rightarrow y\uparrow$

Figure 2.2: PRS for NAND and C-element gates. $y\downarrow$ is shorthand for y:=**false**, and $y\uparrow$ is shorthand for y:=**true**.

[1]We assume that at any moment, the value of a node is either **false** (0) or **true** (1).

2.2.1 Abstract Semantics of PRS

There are a number of possible semantics for PRS. Alain Martin suggests two types of operational models[1]. One of these is the very specific model in which the executions are sequences obtained through weakly-fair selection; this is the model which I call atomic. The other is a more abstract class of models in which the executions are **concurrent composition**s of local sequences. This latter model class defines a (valid) execution as any set satisfying a number of properties[1], which I state as follows:

1. An execution is a set of **events**. Each (nonvacuous) event is the (nonvacuous) firing of a PR (production rule). The firings of a single PR form a sequence. An **execution** is some sort of **concurrent composition** of PR-firing–sequences.

2. The **system state** is an assignment of a value to each node.[2] The effect of an event is to change the value of the target node of the PR to the target value of the PR; these are the only changes to values allowed, i.e., after a PR firing, the system state is the same as it was before the event executed, except for the effect of that PR.

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[3].

To make these semantics rigorous, we must specify the type of concurrent composition that we are using. In Section 2.4 we demonstrate why this choice matters, and in Section 2.4.4 we complete our semantics by choosing a concurrent composition that yields an atomic model.

[2]In the original formulation[1], the system state is not necessarily globally defined before every PR firing. However, the state must be defined at enough nodes that the PR's guard can be evaluated to **true** before the PR nonvacuously fires. Fortunately, there is no loss of generality in assuming that there is always a global state, as we simply ignore the part of the state that is not evaluated.

[3]The original formulation[1] allowed additional, vacuous events of the "vacuous" and "guard not enabled" varieties, but then ignored them in analysis. Ignoring the vacuous events is equivalent to removing them from the model entirely. Therefore, to simplify analysis, we have removed vacuous events from the model.

2.2.2 Stability and Non-Interference (SNI)

Any semantics must admit the following notions of stability and noninterference[1]:

1. An execution is **stable** if a PR is never disabled, except through the firing of that PR[4].

2. An execution is **noninterfering** if for each gate with guard g and opposing guard q (i.e., the other guard of that gate, which pulls in the other direction) the invariant $\neg(g \wedge q)$ 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, i.e., we assume the PRS is **Stable and Non-Interfering (SNI)**.

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

After decomposition, a design is expressed as a set of CHP processes typically of the **buffer** form, written in CHP as $*[L; R]$. 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 CHP notation[1]:

Expression	**Meaning**
L	communication on channel L
L (example) Le (example)	wires implementing channel L (unless otherwise noted, initial wire value is 0).
$[p]$	Wait for condition p to hold.
$[Re \wedge L]$ (example)	Wait for condition $[Re \wedge L]$ to hold.
$*[\texttt{Body}]$	Repeat `Body` forever.
$S_1; S_2$	Perform S_1 and then S_2, sequentially.
$Le\uparrow$	Assign value of 1 to wire Le.
$R\downarrow$	Assign value of 0 to wire R.

Figure 2.3: Expressions used in CHP and HSE.

[4]The original definition of PR stability is that whenever the guard g holds and the rule is (nonvacuously) enabled, then g continues to hold until the rule executes[1]. I prefer the shorter version given above.

Each CHP process is compiled into a **HandShaking Expansion (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:

$$\begin{array}{ll}
\text{CHP Process} & *[\, L \qquad\qquad ; R \qquad\qquad] \\
\multicolumn{2}{l}{\downarrow \text{ compile (communications} \rightarrow \text{DI handshakes)}} \\
\text{HSE Process} & *[\, Le{\uparrow}; [L]; Le{\downarrow}; [\neg L]\, ; [Re]; R{\uparrow}; [\neg Re]; R{\downarrow} \;\;] \\
\multicolumn{2}{l}{\downarrow \text{ reshuffle}} \\
\text{HSE Process } \{Le{\uparrow}\}; & *[\;\; [Re \wedge L]; R{\uparrow}; Le{\downarrow}; [\neg Re \wedge \neg L]; R{\downarrow}; Le{\uparrow} \;\;] \\
\multicolumn{2}{l}{\downarrow \text{ add bubble and state variables}} \\
\begin{array}{l}\text{Implementable} \\ \text{HSE Process}\end{array} & \ldots; *\, [\, [Re \wedge L]; \_R{\downarrow}; Le{\downarrow}; [\neg Re \wedge \neg L]; \_R{\uparrow}; Le{\uparrow}\,]
\end{array}$$

Figure 2.4: 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:

$$*[[Re \wedge L]; \_R{\downarrow}; Le{\downarrow}; [\neg Re \wedge \neg L]; \_R{\uparrow}; Le{\uparrow}]. \tag{2.2}$$

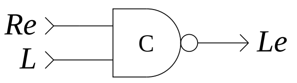
Such a PRS can be found using the following procedure[1]:

1. Assume the hypothesis that the HSE holds. Then Compute the values of all output variables at each semicolon using this assumption.

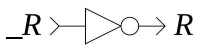
2. Ensure that when a PR's assignment appears in the HSE, the PR is enabled.

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[5]:



$$\begin{array}{ccccc} Re & \wedge & L & \rightarrow & Le\downarrow \\ \neg Re & \wedge & \neg L & \rightarrow & Le\uparrow \end{array}$$



$$\begin{array}{ccc} \_R & = & Le \\ \_R & \rightarrow & R\downarrow \\ \neg\_R & \rightarrow & R\uparrow \end{array}$$

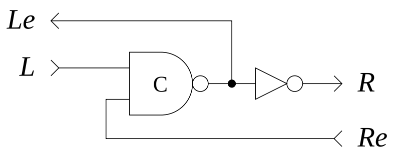


Figure 2.5: Martin-Synthesized PRS for the $*[L; R]$ buffer derived in Figure 2.4.

The HSE shown in (2.2) is known as the **Weak Condition Half Buffer (WCHB)** reshuffling. The circuit shown in Figure 2.5 is a well-known example of a **WCHB buffer**[13][22], also known as a **Muller buffer**. It is the same circuit as shown in Figure 1.2, except that the data is unary instead of binary, for simplicity.

2.4 The Importance of Atomicity

Recall (Section 1.3) that a PRS semantic model is **atomic** if it is equivalent to the weakly-fair sequential selection model. In that model, each PR execution is represented as a single event: each element of the execution sequence is an entire PR firing.

However, recall also (Section 2.2.1) that there are models in which the executions are **concurrent compositions**, and we have asserted (Section 1.3) that such models may or may not be atomic. In this section we use a real circuit example to illustrate this claim, though we save the discussion of concurrent composition for Appendix A.1, as it is not directly relevant to the main result.

[5]In Figure 2.5, the connection notation $\_Re = Le$ means that $\_Re$ is defined to be the same wire as Le.

2.4.1 Feedback Example

The PRS buffer implementation shown in Figure 2.5 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:

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

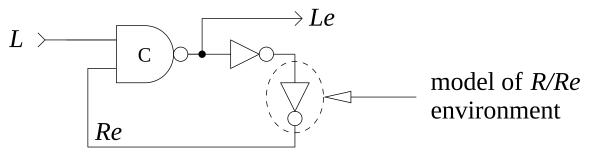


Figure 2.6: Circuit vulnerable to slow transitions on L.

To be correct, the circuit must behave according to the HSE. In this case, the HSE predicts one output transition on Le per input transition on L (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.

2.4.2 Spurious Ring Oscillator

The analog CMOS implementation (see Sections 2.5-2.5.6) of the circuit shown in Figure 2.6 is shown at left in the following figure (we have renamed the nodes without changing the circuit):

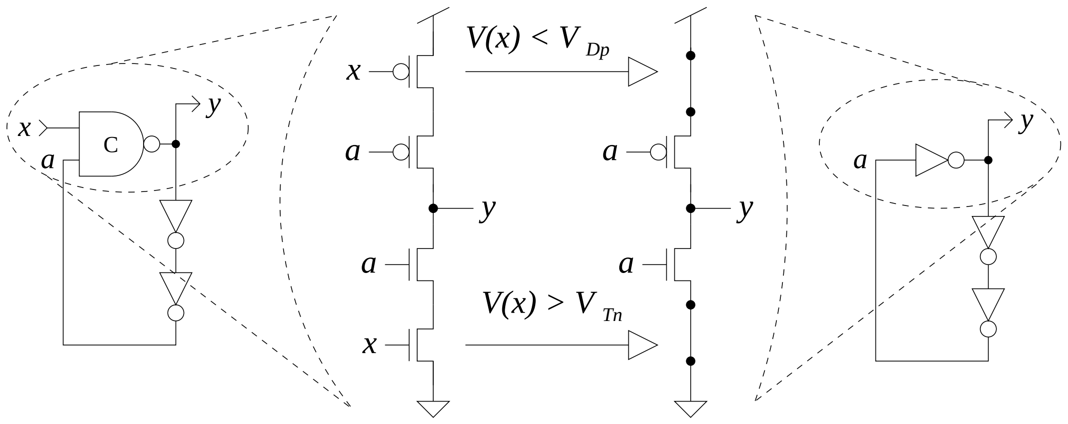


Figure 2.7: When x has an intermediate analog value it enables N and P transistors simultaneously. This leads to ring-oscillator behavior not possible in a digital, atomic model.

We define a **transition** on x as the period of time that 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 is implemented as shown above, then it behaves as an inverter during the transition on x.

If x rises very slowly, then the **slewtime** (i.e., the duration of the transition) is very long. During this long period of time, the virtual inverter is in a loop with the other two inverters, forming a free-running ring oscillator (see waveforms in Section A.1.1). 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 2.6 predicts just one. This example illustrates that slewtimes must be restricted to achieve specified behavior[38][7][39].

2.4.3 Atomic Semantics of PRS

As noted in Section 2.1, a PRS constructed through Martin Synthesis does not come with timing assumptions. But we have just seen that slewtime restrictions must be made in order for such a PRS to behave correctly. The weakest possible static, data-independent restriction is that each slewtime at the input of a gate must be less than all delays between different inputs of that gate[7]. However, this restriction alone does not lead to the simplest possible model.

For the simplest model in which synthesized PRS is correct, we assume an **atomic model** in which all slewtimes are zero, i.e., we assume that each PR firing is a *single action in the execution.* The simplest such model is the **sequential model** – a model (of the form discussed in Section 2.2.1) in which the executions are all *sequences* of PRs satisfying safety and progress[1].

2.4.4 Atomic Timestamp Semantics

While the sequential model is the simplest one to state, it has the problematic restriction that events cannot occur simultaneously even though they theoretically do so 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.

To solve this problem, we can extend our atomic model of PRS (without loss of semantic generality[7]) by adding physical timestamps:

Definition 2 (Calendar) *For any PRS, with* PR*s* $\mathcal{R}$*, any set of* (PR, timestamp) *pairs, i.e., any subset of* $\mathcal{R} \times \mathbf{R}^{\geq 0}$*, is said to be a* **calendar** *when the following two properties hold:*

1. *The set of timestamps can be sorted into a sequence (I.e., this set has no limits other than – possibly – infinity).*
2. *If two different elements* (r, t) *and* (r', t) *have the same timestamp, then* r *and* r' *have different target nodes.*

The elements of a calendar are called **events**, or **PR firings**. The first condition in Definition 2 ensures that the firings of each PR form a sequence (a requirement mentioned in Section 2.2.1) and is needed to rule out non-physical executions such as Zeno's paradox[7][6]. Both conditions are needed in order to complete our definition of the **system state** that we introduced in Section 2.2.1:

Definition 3 (System State) *The* **state** *of a node* y *at time* t *is the target value of the latest event targeting* y *before time* t *or – if there is no such event – the initial value of* y*.*

[6]In Zeno's paradox, an infinite number of assignments can occur before an assignment S. This would make it impossible to evaluate the system state at the time of S. In other incarnations, spurious failures occur: S_0 occured because S_1 occured because S_2 occured ... (we never get to the bottom of the explanation).

Using Definitions 2-3, liveness and safety (first defined in Section 2.2.1) now have natural, rigorous definitions in the atomic timestamp model:

Definition 4 (Liveness) *A calendar is* **live** *if and only if no rule is indefinitely enabled, i.e., there is no rule r and time t after which the system state always enables r*[7]*.*

Definition 5 (Safety) *A calendar is* **safe** *if and only if each event, with timestamp t, is enabled at time t. A rule is enabled if and only if the system state satisfies its guard but has not yet assigned the target node to the target value.*

Definition 6 (Execution) *An* **execution** *is any live, safe calendar.*

We refer to the above definitions collectively as the **atomic timestamp model**. As noted above, this is the only atomic model we need, so henceforth we will refer to it as simply *the* **atomic model**.

2.5 CMOS

CMOS technology provides two complementary devices consisting of both polarities of the MOSFET transistor. The NMOS type of transistor conducts when its **gate terminal** G[8] is at a higher voltage than its other terminals, while the PMOS conducts when its G is lower than its other terminals. Having both types of devices allows circuits to be built that have zero idle power dissipation, in contrast to earlier circuit technologies[34]. The following symbols are used in CMOS circuit diagrams:

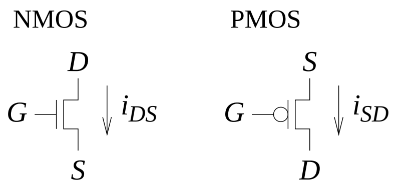


Figure 2.8: CMOS transistors.

[7]Recall that executing a rule always disables the rule, by our definition of (nonvacuous) enabledness.

[8]It is unfortunate that the term **gate** is used to describe both the transistor's gate terminal and the logic gate. It should always be clear which is meant (from either context or direct qualification).

In both types of MOSFETs, current flows between the **source terminal** S and **drain terminal** D when the device is conducting. In contrast to other technologies, MOSFETs have no gate current, so all current can be expressed as the single quantity

$$i_{DS} = -i_{SD}. \tag{2.3}$$

As suggested in Figure 2.8, current usually flows from D to S in an NMOS transistor and from S to D in a PMOS transistor. However, this is only a guideline, and as we will see shortly, the current actually flows from the higher-voltage terminal to the lower-voltage terminal regardless of which of those terminals is S and which of them is D.

2.5.1 NMOS Element-Equation

The element-equation for an NMOS transistor is as follows:

$$i_{DS} = k_n \cdot \Big[\max(v_{GS} - v_{Tn}, 0)^2 - \max(v_{GD} - v_{Tn}, 0)^2\Big]. \tag{2.4}$$

This is the standard model used in SPICE level 0, also known as the **Sau model**[34][9][10]. The **transistor threshold parameter** v_{Tn} determines when the transistor begins conducting, and the **transconductance parameter** k_n determines the rate of conductance increase as the gate voltage increases.

2.5.2 PMOS Element-Equation

A PMOS transistor behaves exactly as an NMOS transistor would behave if all circuit voltages and currents were negated:[11]

$$i_{SD} = k_p \cdot \Big[\max(v_{SG} - v_{Tp}, 0)^2 - \max(v_{DG} - v_{Tp}, 0)^2\Big]. \tag{2.5}$$

[9]In many texts (including the above reference) this equation is presented as four separate cases: cut-off, forward saturation, reverse saturation, and active. I prefer the above form because it is shorter and does not hide the fact that MOSFETs are symmetrical: their behavior is not affected when D and S are swapped. The disadvantage of the above form is that the linear-gain region is hidden; it occurs when both quadratic terms are enabled, leaving a linear residue.

[10]Many texts use a k_n whose value is twice the value used here.

[11]In some texts the parameters k_p and V_{Tp} are also negated, but that convention leads to the use of confusing absolute-value signs whenever a property applies to both types of transistors.

2.5.3 SMO

The element-equations for MOSFETs contain multiple instances of the expression

$$\mathrm{SM0}(x) \overset{\mathrm{def}}{=} \max(x,0)^2, \tag{2.6}$$

which we now add to our vocabulary of primitive operators. We henceforth view the gate current as a simple combination of SM0ed terms as follows:

$$\begin{aligned} i_{DSn}/k_n &= \mathrm{SM0}(v_{GS} - v_{Tn}) - \mathrm{SM0}(v_{GD} - v_{Tn}) \qquad \text{and} \\ i_{SDp}/k_p &= \mathrm{SM0}(v_{SG} - v_{Tp}) - \mathrm{SM0}(v_{DG} - v_{Tp}). \end{aligned} \tag{2.7}$$

2.5.4 Combinational Logic Gates in CMOS

A standard combinational CMOS gate has an output which is always either pulled (by some conductive path) to the **logic 0** supply voltage GND ($\overset{\mathrm{def}}{=}$ 0 volts) or to the **logic 1** supply voltage V_{DD}. The simplest combinational gate is the inverter, whose output y is low (logic 0) whenever the input a is high (logic 1) and high whenever the input is low:

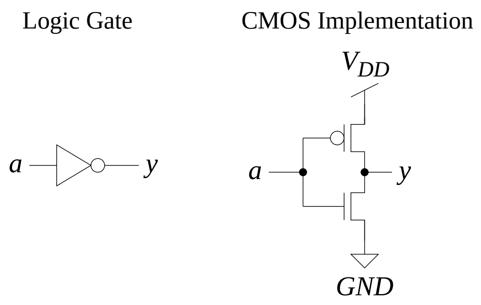


Figure 2.9: CMOS inverter.

Additional characteristics of the CMOS inverter are discussed in Appendix B. The more versatile 2-input NAND gate has the following circuit:

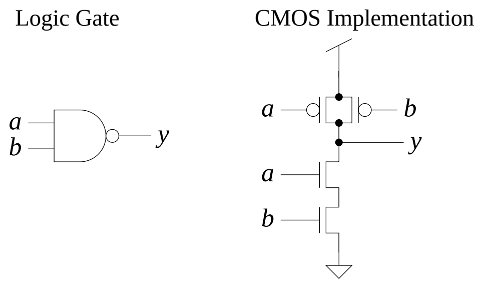


Figure 2.10: CMOS 2-input NAND gate.

The PMOS transistors are wired in parallel, so if either input is low then the output is pulled high; otherwise both inputs must be high, and the output is pulled low through the NMOS **series chain**. Notice that there is never a direct path from V_{DD} to GND; some transistor is always off, except possibly during input transitions.

2.5.5 Dynamic Logic Gates in CMOS

In combinational gates, for every possible combination of inputs the output is directly driven to either V_{DD} or GND. Such a gate cannot, on its own, remember the values of previous inputs. In contrast, a **dynamic gate** (a.k.a. a **state-holding gate** or **sequential gate**) remembers its latest value for some input combinations[12]. The circuits in this thesis involve only one such gate, the 2-input C-element[13]:

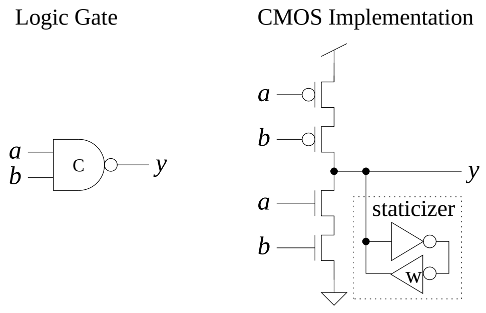


Figure 2.11: CMOS 2-input C-element.

[12]In some texts, a gate is only called "dynamic" if it obeys the above definition *and* has no staticizer. Thus, those texts would *not* consider an SRAM cell to be dynamic, because it has a staticizer. In this thesis our dynamic gates can have staticizers: in fact, they must, as we do not deal with circuits that require periodic refreshing.

The output holds its value when it is not driven because of charge stored in its lumped capacitance (see Section 3.2.1). To prevent this charge from slowly leaking, we always add a circuit with a weak driver called a **staticizer**, shown above.[1]

While many of our example circuits use staticizers, they are in fact not necessary for the implementation of arbitrary computation, as any dynamic gate can be implemented as a network of combinational gates.[1] Therefore the main theoretical result of this thesis does not rely on the existence of staticizers. Nonetheless we consider staticizers, because of their popularity in practice.

2.5.6 Canonical CMOS Implementation of PRS

There is a general canonical method to construct a circuit for an arbitrary PRS[1]. Recall (Section 2.2) that a general gate in PRS consists of two PRs, $g \to y\downarrow$ and $q \to y\uparrow$.

The **pulldown guard** g is implemented as a **PullDown Network (PDN)**, designed to conduct whenever g holds, and a **PullUp Network (PUN)**, designed to conduct whenever q holds:

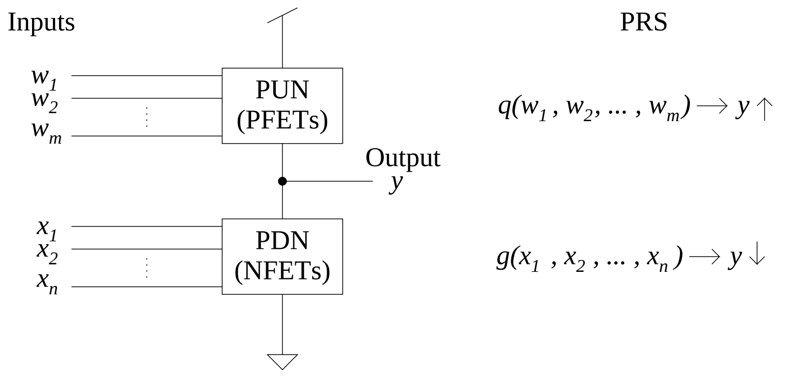

Inputs
PRS
w_1
w_2
w_m
PUN (PFETs)
$q(w_1, w_2, \ldots, w_m) \longrightarrow y\uparrow$
Output y
x_1
x_2
x_n
PDN (NFETs)
$g(x_1, x_2, \ldots, x_n) \longrightarrow y\downarrow$

Figure 2.12: The CMOS-gate implementation of a general pair of PRs.

Such networks can be found for any desired guard function[51]. For reasons explained in Appendix B.1, PUNs are implemented only in p-transistors and PDNs are implemented only in n-transistors[1].

2.6 Rigorous Verification: Synchronous versus Asynchronous

Several standard concepts in rigorous synchronous analog verification have direct analogs in asynchronous verification. To see this, we will discuss verification of synchronous circuits in this section.

Rigorous verification of synchronous circuits has a simpler theory than that of asynchronous circuits because the signal bounds are derived directly from the clock and typically have a simple form. For example, for static clocked CMOS we do not have to give any characterization of circuit values before the values have reached steady-state.

In fact, for static clocked circuits we *cannot* give a complete dynamic characterization because such circuits function perfectly well with glitchy signals (i.e., signals that are temporarily allowed to oscillate uncontrollably). High-performance dynamic clocked circuits involve additional keepout regions that prevent glitches, but these are simple asymmetric-pulldown extensions of the combinational theory. Therefore we discuss the basic method used for combinational logic.

2.6.1 Steady-State Methods for Combinational Logic

The verification of standard combinational logic begins with the designation of $valid(0)$ and $valid(1)$ regions within the space of possible signal voltages. As shown below, the remaining space is known as the **forbidden zone**[11]:

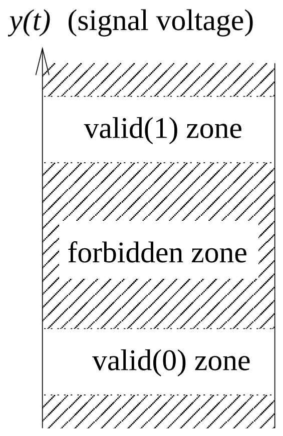


Figure 2.13: Valid steady-state regions for the analog voltage-signal $y(t)$. The forbidden zone is shown as cross-hatched.

Each voltage-signal $y(t)$ converges to a valid representation of a digital value $H(y)$. More precisely, we have the following property:

$$converges\Big(y, H(y)\Big) \overset{\text{def}}{=} \exists t : \ \forall s {\geq} t : \ y(t) {\in} valid\Big(H(y)\Big) \tag{2.8}$$

2.6.2 Induction

Analog verification of combinational logic is achieved when $converges(y, H(y))$ holds for all y in the circuit. For an acyclic circuit this is proved by induction on the circuit graph. We assume the **base case (input specification)** that $converges(a, H(a))$ holds for all inputs a and must show the **inductive step (analog gate-specification)** that $converges(y, H(y))$ holds for each logic-gate output y, assuming $converges(x, H(x))$ holds for each input x to that gate.

For example, consider a circuit with three inputs a, b, and c, shown below:

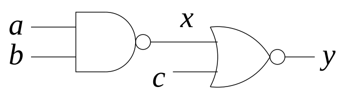


Figure 2.14: A combinational logic circuit.

The static digital specification for this circuit is

$$\begin{aligned} H(x) &= \text{NAND}\Big(H(a), H(b)\Big) \qquad \text{and} \\ H(y) &= \text{NOR}\Big(H(x), H(c)\Big). \end{aligned} \tag{2.9}$$

By induction, therefore, we can obtain a rigorous analog verification by demonstrating the following two (analog) gate specifications:

$$\begin{aligned} \forall A, B : \; converges(a, A) \wedge converges(b, B) &\quad\Rightarrow\quad converges(x, \text{NAND}(A, B)) \\ \forall X, C : \; converges(x, X) \wedge converges(c, C) &\quad\Rightarrow\quad converges(y, \text{NOR}(X, C)) \end{aligned} \tag{2.10}$$

Standard induction (i.e., the form used above) works on acyclic graphs (such as the one above).

Unfortunately, asynchronous circuits are not naturally broken into acyclic components. Even **generalized induction** cannot be directly applied to general cyclic circuits. Generalized induction works on any set with a traversal operator; the property then holds over the closure of the base set under that operator[52]. Unfortunately, the traversal operators appropriate to analog verification depend on all gate inputs, not just on those for which we trivially have a base case. This yields a trivial closure for cyclic circuits. We solve this problem in Section 4.4 by introducing a novel form of induction.

2.6.3 Timing Closure: Clocked Static vs. QDI

Our convergence definition introduced in Section 2.6.1 requires only that a signal **eventually** reach the proper valid region. However, to know how fast to run the clock we need to know **how long** this will take. Thus, in clocked circuits these regions are triggered by the clock. By contrast, for QDI circuits the signals are self-timed, so the regions are somehow derived from the data signals themselves. For example, the relationships between sets of signals and their forbidden zones are compared below for both clocked static and QDI signals:

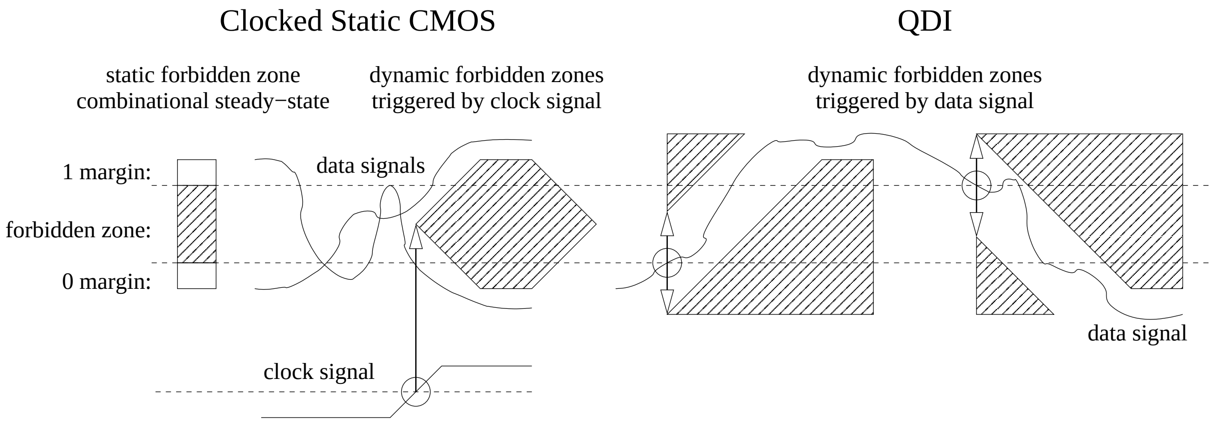


Figure 2.15: Triggering of forbidden-zone onsets in clocked static CMOS and in QDI.

The spacing and extent of the regions in time are determined by some estimate of the circuit's speed, such as logical effort[43]. In clocked circuits these speed estimates go into setting the clock frequency. In QDI circuits the speed estimates are the conditions of my observation theorem, (Chapter 3) and the values are hidden from the digital designer.

2.6.4 Inertial Delay: Clocked Static vs. QDI

With few exceptions,[13] circuit design styles require the circuit designer to prevent **race condition**s in which signals *propagate too quickly.*

All asynchronous circuits are subject to race conditions because they can begin arbitrarily complex computation without external triggering. If all transitions in that computation were to happen at the same time, then all nodes would become undefined[14]. Synchronous

[13]Two-phase nonoverlapping clocks is one exception (perhaps the only one).

[14]Of course, in a partial order model everything cannot occur "at the same time" assuming there is no Zeno's paradox. However, digital models do not (in the absense of the assumptions we are developing) in general describe the behavior of analog circuits, and there *are* analog implementations of any digital QDI circuit that are so fast that all nodes become intermediate-valued shortly after startup. Hopefully the specific

circuits with edge-triggered clocks are also subject to race conditions[11].

Race conditions are avoided by ensuring that signals do not propagate too fast. The lower bound on propagation delay is referred to as an **inertial delay**, or **contamination delay**. As with other timing closure properties, the inertial delay is measured from the clock in clocked circuits and from the signals themselves in self-timed circuits, as shown below:

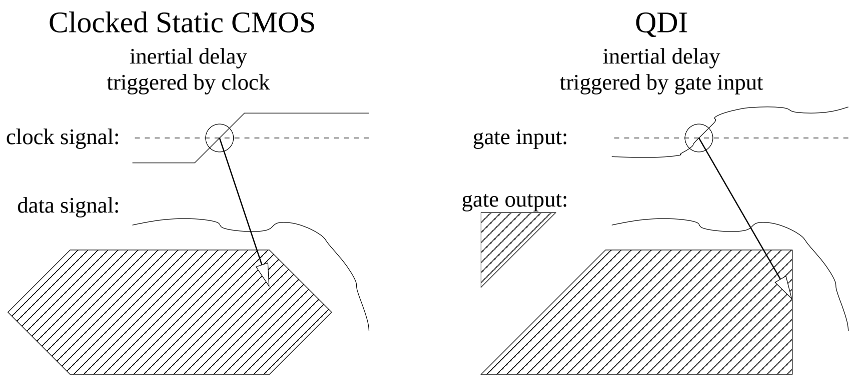


Figure 2.16: Measurement of inertial delay in clocked static CMOS and in QDI.

2.6.5 Limitations of Steady-State Arguments for Async. Verification

Perhaps the most fundamental difference between synchronous and asynchronous analog verification is in the types of properties that are propagated in the inductive proofs. The synchronous proofs for combinational logic rely entirely on steady-state values.

Clearly such arguments are highly inadequate for asynchronous circuits. In fact, any argument which does not specifically bound every transition individually would fail to diagnose catastrophic drifts in the transition shape, as we discuss in Section A.4.

example discussed in Section 2.4.2 is enough to convince the reader that all digital QDI circuits are subject to race conditions.

2.6.6 Summary of Synchronous/Asynchronous Verification Differences

In this section we have drawn several analogies between concepts in synchronous and asynchronous analog verification and identified the following differences:

Concept	**Static Clocked CMOS**	**QDI**
digital specification	clocked combinational	SNI PRS
property propagated by induction	steady-state values	bounded transitions
timing source for signal x	clock signal	signal x itself
forbidden-zone onset trigger for x	clock edge	edge on x
inertial delay measured from	clock signal	gate input
inertial delay measured to	data signal	gate output
inductive assumption	good input value	good input transition
inductive step	good output value	good output transition

Figure 2.17: Differences between synchronous and asynchronous verification.

Chapter 3

The Statement of the Observation Theorem

As discussed in Section 1.4, a PRS implementation can only be considered correct if there is some **observation rule** that yields a correct computation *for any physical evolution* of the system. The **observation theorem** states that for a particular observation rule (defined in Section 3.5), this is the case. By construction of the PRS, a behavior is assumed to be a correct computation if it is any execution in the atomic model discussed in Section 2.4.4.

Can we mathematically define a **physical evolution**? All known practical physical PRS implementations are most accurately described by *analog models*. As discussed in Section 1.1, these models reveal failures not revealed in digital models. The SPICE community believes that analog models are a solid abstraction of physical laws and will continue to be the most accurate available models, so we hop onto the analog bandwagon.

An **analog evolution** of a PRS implementation consists of a time-dependent **signal** (i.e., voltage signal) for each circuit node. We give a complete definition of the analog model in Section 3.2. Like a digital signal, an analog signal generally alternates between 0 and 1. However, the analog signal is continuous, and so it cannot instantaneously change from 0 to 1. As discussed in Section 2.4.1, these changes occur as alternating **transition**s, the time intervals over which the signal alternately rises and falls. For each signal y, our observation rule ϕ must map each analog transition to a digital transition, as shown below:

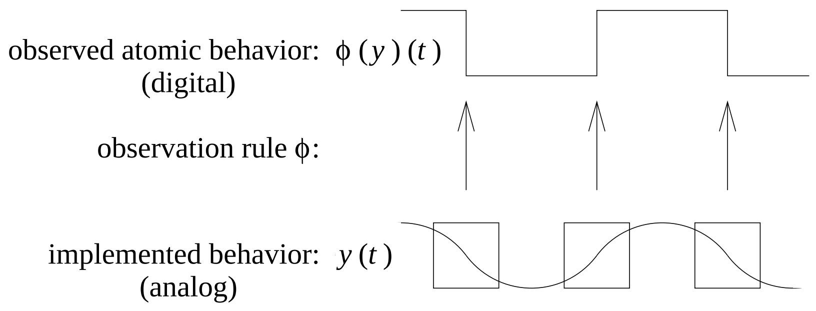

observed atomic behavior: $\phi(y)(t)$
(digital)
observation rule ϕ:
implemented behavior: $y(t)$
(analog)

Figure 3.1: Observation rule ϕ applied to analog signal y gives atomic, digital transitions.

The digital transitions $\phi(y)$ will be represented as events in the atomic timestamp model. By construction, these event sequences will not have Zeno's-paradox limits. Therefore, if L is the complete analog evolution (i.e., a signal for every node), then $\phi(L)$ is a calendar (Section 2.4.4). As noted above, the observation theorem simply states that any analog evolution $\phi(L)$ is an execution.

Unfortunately, $\phi(L)$ is not always an execution (as discussed in Section 1.1), so it cannot be proved in general that $\phi(L)$ is an execution; certain conditions are necessary. We state these conditions mathematically in Sections 3.6.1-3.6.3. However, we begin by explaining how these conditions are obtained for a general circuit.

3.1 User's Guide

We now describe how to attempt verification of any canonical implementation (Section 2.5.6) of any SNI PRS (Section 2.2.2). In Section 3.2.1 we present a standard DE (Differential Equation) form which describes any completely sized and extracted PRS implementation to first order. No such equation can be completely accurate, so in Section 3.3 we introduce a noise-bound ηwhich allows us to make our description accurate.

After complete sizing of the circuit, the user then selects a handful of thresholds for each node. The circuit and these thresholds are fed into FenceCalc™(see Section 6). If there exist partial fences satisfying the necessary conditions, then FenceCalc™ will find them and report success. Recall (Section 1.8) that there are two types of partial fences, transient and DC. The observation theorem makes assumptions about both the transient and DC fences, which we state in Sections 3.6.1 and 3.6.2, respectively. If the conditions are not met, we meet them by adding delay to the circuit as discussed in Section 7.6.

If these conditions are satisfied, then one can conclude that the observation of any analog evolution L having the appropriate initial conditions (see Section 3.4) and within noise bound η of satisfying the DE is an atomic execution, as illustrated below:

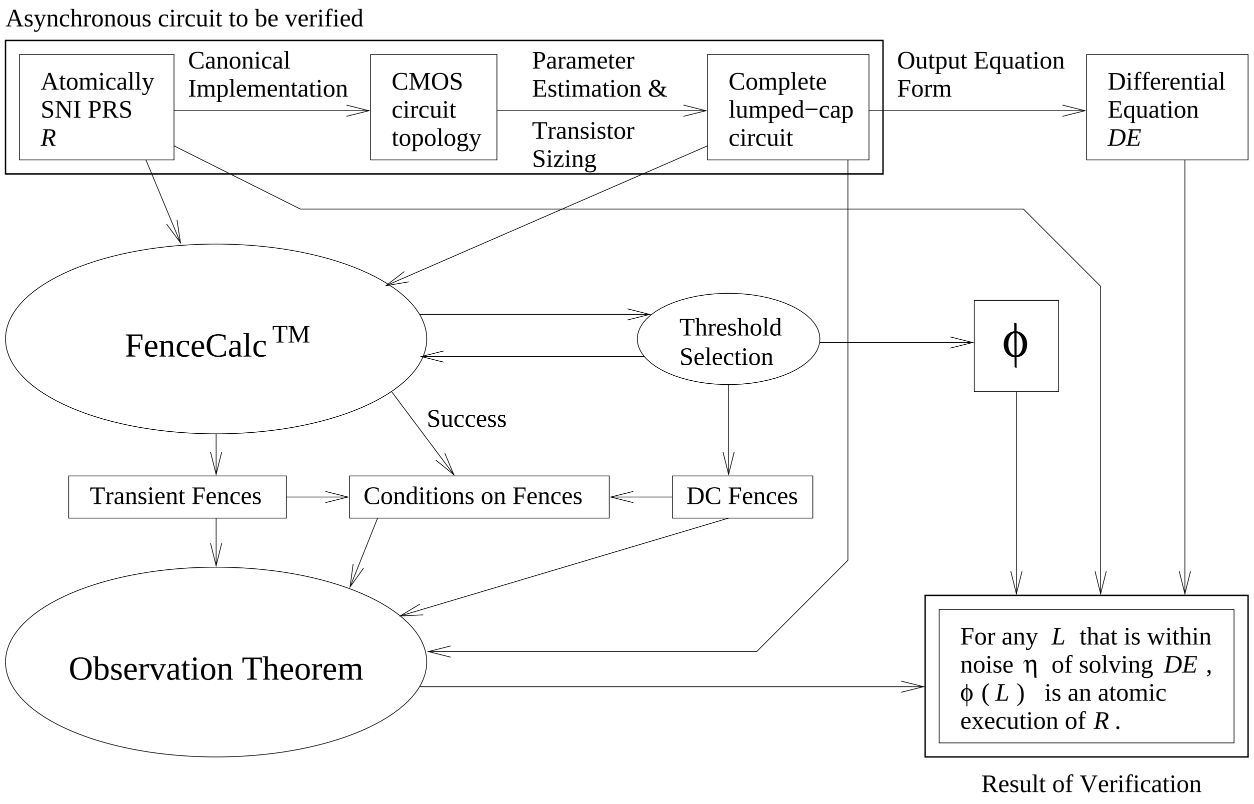


Figure 3.2: How to verify a circuit using the observation theorem.

The circuit is then rigorously verified.

3.2 The Analog Model

3.2.1 Lumped Unit-Capacitance Model

Any circuit that is capable of independent computation (or any repeated activity that does not rely on changing input signals) must include some circuit element whose equation involves time variation. Clearly the MOSFET equations introduced in Section 2.5 do not posess this property. In CMOS, the time-varying elements are capacitors. The time-evolution of a circuit can be accurately modeled by adding a capacitor to the output of each gate as follows:

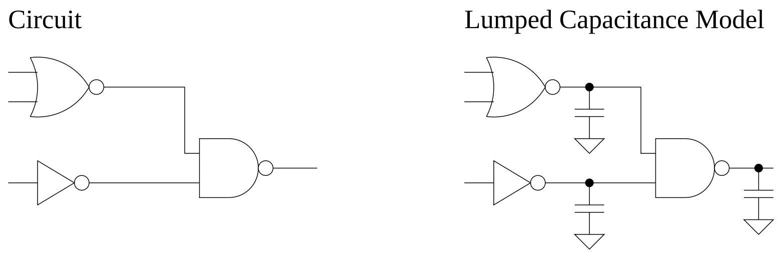


Figure 3.3: Lumped capacitance model of a simple circuit.

The value of a capacitor at a given node is a lump sum of the drain capacitance of all connected transistor inputs, plus the gate capacitance of all connected transistor outputs, plus the capacitance of all connected wiring. While there are other effects such as capacitance between internal nodes in logic gates and wiring resistance and inductance, these are all second-order effects that we will view as noise in Section 3.3.

In the lumped capacitance model, each circuit element is driving exactly one capacitor. Therefore, the behavior of our circuit is equivalent to a modified one in which for each capacitor of capacitance C driven by a transistor of transconductance k we now have a capacitor of capacitance 1 driven by a transistor of **relative transconductance** k/C. This **unit-capacitance convention** eliminates unnecessary parameters and units from our circuit equations.

We view the lumped capacitances as being attached to the outputs, but this does not imply that the outputs are the primary *physical* contributors of lumped capacitance: in many circuits the transistor *inputs* present most of the capacitance. Rather, we view the capacitance as being attached to the output because this simplifies the form of the circuit equations, as I show in the next section.

As an added justification for using relative transconductance, it has been shown that for a large class of performance metrics, a circuit is optimally sized when k is always proportional to C[53], which occurs when the relative transconductance is constant throughout the circuit.

3.2.2 Output Equations: 2-input NAND gate

Consider all currents and voltages in the lumped capacitance circuit for a 2-input NAND gate:

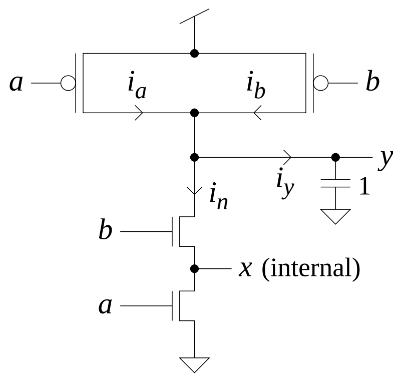


Figure 3.4: Analysis of the 2-input NAND gate.

The only connections to the output y that are not shown in the above figure are the connections to the inputs of the next stage (i.e., the inputs to the circuit that uses the result of our NAND gate). In our model, any capacitance on those inputs is already included in our output capacitor. Therefore the only omitted connections to y are gate terminals, which do not sink any current. The operation of the output thus depends only on the input signals and the internal state of the circuit shown above.

Any electronic circuit is described by a complete system of algebraic and differential equations consisting of the element-equations for all circuit elements, and Kirchoff's laws.[34] We now list all such equations for the circuit shown in Figure 3.4:

Law / Constituent	**Equation**
KVL	All voltage drops can be determined by the voltages a, b, x, y. I.e., $V_{DS,Na} = x, V_{DS,Nb} = y - x$, etc.
KCL	$i_{DS} = i_n$ for both NMOS transistors and $i_y = i_a + i_b - i_n$
Capacitor	$y' = i_y \qquad (\text{i.e., } \frac{\mathrm{d}y}{\mathrm{d}t} = i_y)$
NMOS	$i_n/k_n = \mathrm{SM0}(b - x - V_{Tn}) - \mathrm{SM0}(b - y - V_{Tn})$ $= \mathrm{SM0}(a - V_{Tn}) - \mathrm{SM0}(a - x - V_{Tn})$
PMOS	$i_a/k_p = \mathrm{SM0}(V_{DD} - a - V_{Tp}) - \mathrm{SM0}(y - a - V_{Tp})$ $i_b/k_p = \mathrm{SM0}(V_{DD} - b - V_{Tp}) - \mathrm{SM0}(y - b - V_{Tp})$

Figure 3.5: Complete circuit equations for the circuit of Figure 3.4.

Notice that there are two NMOS equations for the i_n term. These equations actually relate two unknowns: i_n and x. Therefore one would expect x to be implicitly determined unless the system is coincidentally underconstrained. In Appendix D.2 we will see that this is the case (when the system is underconstrained it does not affect the output current). y' is therefore a function of the input and output voltages. We now argue that this is the case for any other CMOS gate of the type introduced in Section 2.5.6.

3.2.3 Output Equations: Canonical Form

Now consider the currents and voltages in the implementation of an arbitrary 2-input PRS gate:

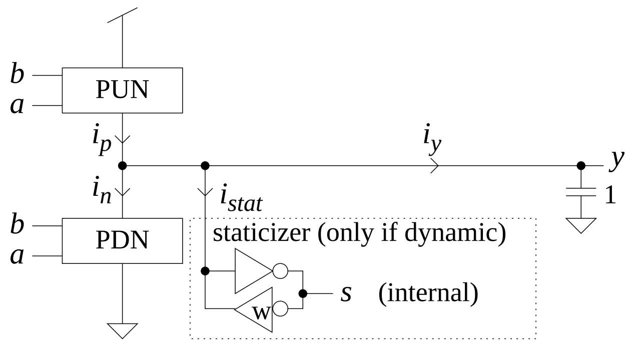


Figure 3.6: Analysis of the implementation of a general 2-input PRS gate.

The current into the capacitor at y is the sum of three terms, defined as follows:

Term	**Meaning**
$i_p(a,b,y)$	PUN source current
$i_n(a,b,y)$	PDN sink current
$i_{stat}(s,y)$	staticizer current

Figure 3.7: Terms of the general output current i_y.

As with the NAND gate example, I show that the PUN and PDN current depends only on a, b, and y, for a general 2-input gate. Actually, for a dynamic gate the staticizer has an internal node s, which the output additionally depends on. The general form of y' therefore depends on a, b, s, and y:

$$y' = i_y(a,b,s,y) = i_p(a,b,y) - i_n(a,b,y) + i_{stat}(s,y) \tag{3.1}$$

Further analysis is required to characterize s at any given moment, though we can always bound i_{stat} based on the weak relative transconductance of the staticizer's output transistors. If there is no staticizer present, we let $i_{stat} = 0$.

3.2.4 Generalization to Other Circuit Families

In Chapter 4 I prove the observation theorem assuming we are dealing with CMOS circuits, described by the equations presented in Section 3.2.3. Therefore the theorem can be understood as a theorem about CMOS circuits. However, the theorem depends only on specific assumptions about the circuit equations, listed below. Therefore the theorem can be applied to any circuit family satisfying the following conditions:

1. **Restricted composition.** All capacitances can be modeled as either lumped output capacitance or noise. The only circuit elements are transistors and lumped capacitances.

2. **No internal hysteresis.** The values of (i.e., voltages on) all intermediate nodes in a gate (i.e., nodes that are neither input nor output) are implicitly determined by input and output voltages.

3. **Steady-state correctness**. If a gate implements a PR pair, and the input voltages remain in a region associated with the completion of assignment to state s, and the PR pair implies that the output becomes b if the inputs remain s, then the output is bounded by an exponential which decays to the completion threshold V_{bC}.

4. **Monotonicity** of the output current. Any deviation (from monotonicity) must be modelable as noise. This also ensures that our fences are monotonic so we can translate them freely in Section 6.6. For CMOS we make a necessary exception when the output is outside the supply rails.

5. **Lipschitz condition** for the output current. This is needed in Euler error analysis (Section D.5), and we use it to justify our noise bound in Section 3.3. An output current function f is **Lipschitz** if there exists a **Lipschitz constant** K such that for all pairs of output voltages x_1 and x_2, we have

$$|f(t, x_1) - f(t, x_2)| \leq K|x_1 - x_2|. \tag{3.2}$$

   This condition is automatically satisfied if $\delta f/\delta x$ is continuous[8], as in CMOS. It neither implies nor is implied by monotonicity.

6. **Slewtime Restoration**. This property guarantees that given a long enough chain of delay elements, a fence pair of any slewtime is mapped to a fence pair of a fast slewtime. We will make this condition precise when we discuss the addition of chains of delay elements in Section 7.6.

These conditions on a circuit family guarantee that we can prove the correctness of the implementation of any PRS with sufficiently long feedback cycles. The latter condition must be checked after threshold parameters are chosen, as we discuss in Section 3.6.

3.3 The Noise Model

The physical trajectories of parts of real machines never *precisely* satisfy equations, because there are always noise sources that cannot be controlled. In the case of CMOS circuits, there are many other reasons why the system does not satisfy Equation 3.1 precisely. The parameters of CMOS circuits are never known precisely, owing to manufacturing tolerances worse than 10%, statistical doping, etc.

More importantly, we are using a model that is known to be merely a first-order model where specific second-order corrections have been identified. As noted, the lumped capacitance model is already an approximation to a model in which there are other capacitances. It is also known that there are **leakage currents**: small charges continuously leak into the substrate regardless of the states of the transistors, and a small **subthreshold current** flows through each transistor in the cut off state. Finally, there are other second-order effects due to the spatial nonuniformity of the device and the effect of substrate bias[54].

We define **noise** as any effect which causes the actual behavior to differ from Equation 3.1. Each such effect (including any of the ones we just discussed) can be classified into the four categories listed below.

1. **Current noise on an output**. The total current i_y into a lumped capacitor differs from Equation 3.1 in absolute value by some η. For example, there is leakage.

2. **Voltage noise on an output**. The output differs from $\int i_y(t)\mathrm{d}t$ by some $v_{\text{noise}}(t)$. For example, there could be an IR drop across an unwanted resistor.

3. **Current noise on an input**. An input consumes nonzero current (other than the lumped capacitance which is already modeled), e.g., because of gate leakage.

4. **Voltage noise on an input**. The voltages $a(t)$ and $b(t)$ (inputs to Equation 3.1) are to be substituted with noisy versions of these voltages.

All non-external inputs are connected to outputs. By KCL, therefore, we can view input current noise (3) as a component of the output current noise (1) for the output connected to it. Current noise on external inputs can be ignored because we will define the environment specification in terms of input *voltages*. Similarly, voltage noise on an output (2) can be considered as a component of (4), voltage noise on all connected inputs (but not the other way around).

This leaves us with (1) and (4). We prefer form (1) because it is clearly the most general form: A voltage noise on the input can be viewed as a current noise on the output, with the Lipschitz constant as a bound on the ratio between them. On the other hand, a current noise cannot be effectively modeled as a voltage noise. For example, consider a (flawed) circuit with dynamic gates whose staticizers are missing. The output voltages can drift by a (practically) arbitrary amount, leading to a very large $v_{\text{noise}}(t)$ even if the actual leakage rate is very low.

The output current noise η is easily incorporated into Equation 3.1 by turning that equation into an inequality:

$$|y' - i_y(a, b, s, y)| \le \eta \tag{3.3}$$

Finally, as we show in Section D.5, this is also the form of the standard bound on Euler error, so we can view η as a sum of individual ηs for computational, physical, and modelling error.

Henceforth we refer to any solution to Inequality 3.3 as an **analog evolution**, provided it satisfies the proper initial analog conditions.

3.4 Initial Analog Conditions

Any circuit capable of independent computation has an internal state which must be reset properly. In our model the internal state is completely represented by the voltage signals of all nodes. During reset these signals are analyzed as in a standard combinational circuit, and it is shown that they all reach stable values.[1]An analog signal is considered stable if it remains in the valid 0 or 1 zone[11] for that signal.

We will analyze two types of circuits:

1. Independently-operating circuits which have a special reset procedure and
2. Circuits which wait for data from their environment before doing anything after reset.

For both types of circuits we define $t = 0$ as the time when the circuit has been reset and is ready to compute. For reasons to become clear in Sections 3.5-3.6, we define the valid regions for the second type of circuit using the constants $DC_{y\updownarrow}$ and $V_{y\updownarrow C}$, as follows:

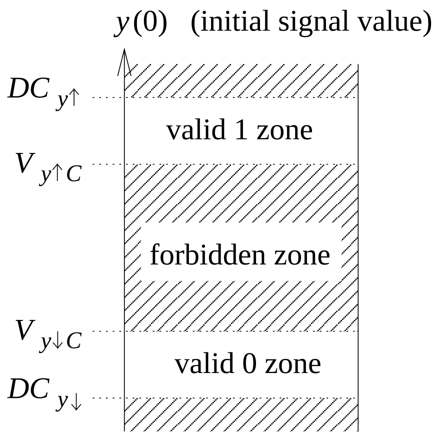


Figure 3.8: Valid initial conditions for the signal $y(t)$.

[1]Reaching the known reset state may require several environment transitions and (for some circuits) additional *reset circuitry*.[1] Properly designed reset circuitry is active only during reset, and after reset it can be modeled as additional capacitance.

3.4.1 Circuits with Initially-Enabled Internal PRs

Our basic observation theorem assumes that the only initially-enabled PRs are the PRs describing the environment. However, in our synchronized–ring-pair circuit example the circuit operates independently of the environment (in fact, there is no environment), so there are initially-enabled internal PRs.

There are two choices for how our result can be applied to such a circuit:

1. Use the initial conditions developed above. The guard of an initially enabled rule $y\downarrow$ has some enabled disjunct, all of whose terms are in the Valid 1 region. Since the solution to a DE does not depend on single points, we can assume that the inputs jumped from Valid 0 to Valid 1 just after $t = 0$. The fences for these terms must be expanded to include this initial "transition". Then the basic version of the observation theorem can be applied.

2. Force the terms of the initially-enabled guard into a transition allowed by its fence hypotheses. We use this approach in our example because it has the advantage of removing troublesome singularities from the fences (which are unrelated to the fundamental question of whether the circuit operates). The disadvantage is that a special reset circuit must be designed to force the initially-enabled target node into a controlled release just after reset.

Technically, to use the second choice we would need to modify our circuit model to include the forcing term in order to achieve the desired startup behavior. Fortunately, however, our main example (the WCHB buffer chain) does not have initially-enabled PRs except in the environment, so we make no modifications to the theorem for now.

3.5 The Four-Threshold Automaton

In this section we show that four thresholds are necessary and sufficient to define an observation rule. We define a particular four-threshold automaton which suffices to prove the theorem, and we explain why the theorem cannot be proved with fewer thresholds.

The simplest way to digitally observe an analog signal is to define some constant threshold voltage V and to say that $y = 1$ digitally whenenver $y > V$. There are several problems with such an observation rule. First, if the signal is close to V, a small noise on that signal can be erroneously interpreted as several transitions. Second, to prove an observation theorem it helps[7] to be able to derive useful information about a signal based on its observation. With the preceding rule we might know a signal's digital value but we can never use this information to conclude that the signal is very far from V.

As a first improvement on our single-threshold rule we can use two constant threshold voltage parameters, much as a Schmitt trigger does[34][54]. We refer to these thresholds as **completion thresholds** and denote them as follows:

Symbol	**Name of Parameter**
$V_{y\uparrow C}$	$y_\uparrow$ completion threshold
$V_{y\downarrow C}$	$y_\downarrow$ completion threshold

Figure 3.9: Completion thresholds for y.

A signal y (initially 0) is considered 0 until it reaches $V_{y\uparrow C}$; then it is considered 1 until it subsequently reaches $V_{y\downarrow C}$.

While the two-threshold rule is a vast improvement over the single-threshold rule, we unfortunately cannot directly use it as an observation rule. The problem is that these thresholds do not report an event until the signal has **completed** most of its transition (hence the name "completion thresholds"). Unfortunately, the transition can begin propagating long before this happens. Thus any two-threshold rule has the flaw that the consequence of propagation can be reported before the cause (for a large important class of verifiable circuits).

Owing to this problem, we must additionally define **initiation**, the time when a transition can begin propagating to the next logic stage, which clearly is before completion. Only an observation rule based upon *initiation* events yields valid atomic exections[38][7]. Therefore we introduce **initiation thresholds**, denoted as follows:

Symbol	**Name of Parameter**
$V_{y\uparrow I}$	$y_\uparrow$ initiation threshold
$V_{y\downarrow I}$	$y_\downarrow$ initiation threshold

Figure 3.10: Initiation thresholds for y.

We refer to an automaton based on the thresholds $V_{y\updownarrow I}$ and $V_{y\updownarrow C}$ as a **four-threshold automaton**. Specifically, the output of the automaton Φ given a signal y is a (usually infinite) sequence of timestamps

$$\Phi(y) = \Big\{(t_{y\uparrow I})_1,\ (t_{y\uparrow C})_1,\ (t_{y\downarrow I})_2,\ (t_{y\downarrow C})_2,\ (t_{y\uparrow I})_3,\ (t_{y\uparrow C})_3,\ \ldots\Big\}, \tag{3.4}$$

which indicate when the corresponding thresholds were crossed, as shown below:

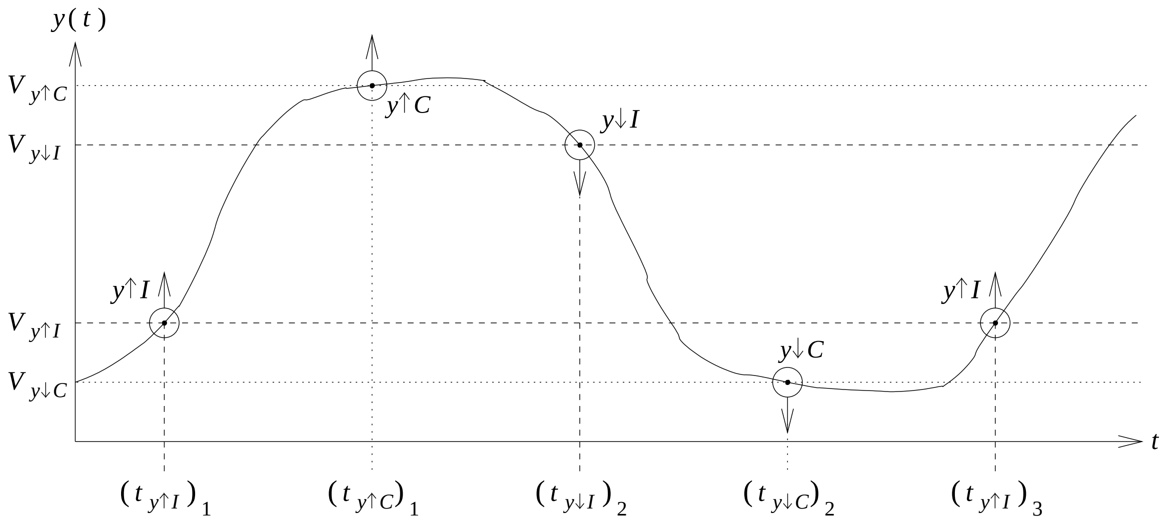
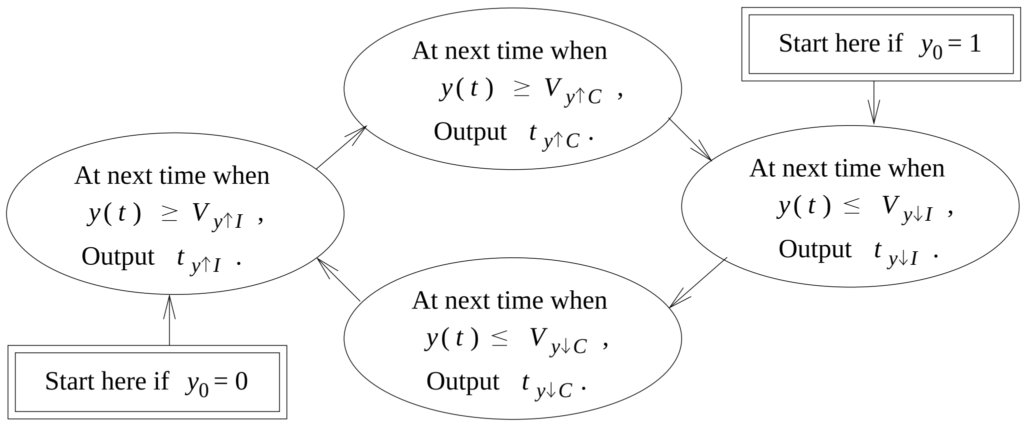


Figure 3.11: Sample operation of the four-threshold automaton.

For an initial digital value of 1, the sequence is similar but the $\updownarrow$ are flipped. Events with even indices assign the reset value; events with odd indices assign the opposite value.

Mathematically, the automaton is defined by induction. Each output in the sequence is based on the previous output and the time at which some threshold is crossed after that:



Start here if $y_0 = 1$
At next time when $y(t) \geq V_{y\uparrow C}$, Output $t_{y\uparrow C}$.
At next time when $y(t) \leq V_{y\downarrow I}$, Output $t_{y\downarrow I}$.
At next time when $y(t) \geq V_{y\uparrow I}$, Output $t_{y\uparrow I}$.
At next time when $y(t) \leq V_{y\downarrow C}$, Output $t_{y\downarrow C}$.
Start here if $y_0 = 0$

Figure 3.12: Inductive definition of $\Phi(y)$.

Notice that (like a Schmitt trigger) this construction is designed to ignore small noise, even if that noise results in the signal crossing a threshold multiple times. For example, a $y{\uparrow}I$ event can only be immediately followed by a $y{\uparrow}C$ event, regardless of how many times the signal may cross $V_{y\uparrow I}$.

We let $\phi(y)$ – lowercase ϕ – denote the projection of $\Phi(y)$ that selects only the I (initiation) events. The sequences $\Phi(y)$ and $\phi(y)$ do not converge (no Zeno's paradox) because $\mathrm{d}y(t)/\mathrm{d}t$ is bounded. ϕ is therefore a proper observation rule (i.e., its image is calendars).

3.6 Partial Fences

Recall that in our approach to the observation theorem we begin by proving that each signal $y(t)$ is bounded by lower and upper fences $l_y(t)$ and $u_y(t)$, respectively (Section 1.5). Safety and liveness of the observation are easy to show once this property has been established.

Also recall that the bounds are explicitly representable as piecewise functions[2] that alternate between **transient** (i.e., finite) and **DC** (i.e., constant) portions (Section 1.8) which we call **partial fences**[3]. For example, the fences between the first two transitions of a signal y are as follows, assuming $y\uparrow$ begins at time $t = 0$:

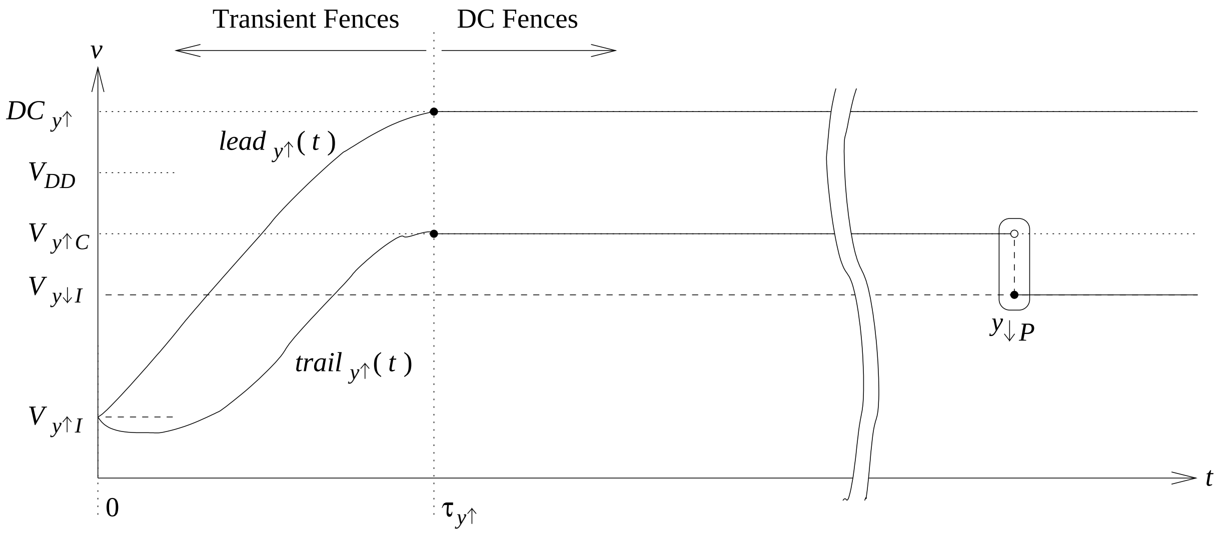


Figure 3.13: Partial fences for $y_\uparrow$.

[2]Recall that fences can be accurate without precisely solving the DE. See Appendix C for simple examples.

[3]We use the term **partial fence** to denote a fence segment of a fixed mathematical form that holds over a specific time interval. As we will see, each partial fence is only *part* of the overall upper or lower bound on a signal.

As suggested in Figure 3.13, we assume that the signal potentially takes on its most extreme value after the transition has completed and the signal is settling into a valid (completed) 0 or 1. Therefore our upper bound on a completed 1 and lower bound on a completed 0 are global bounds that hold for all time, defined as follows:

Symbol	**Name of Parameter**
$DC_{y\uparrow}$	DC upper bound on y
$DC_{y\downarrow}$	DC lower bound on y

Figure 3.14: Indefinite DC bounds on y (holding for all time).

The lower bound on a completed $y\uparrow$ is the completion threshold $V_{y\uparrow C}$. At some point just before the next transition (if any) this bound no longer holds as the signal approaches $V_{y\downarrow I}$. In Section 3.6.1 we will see that this **preparation event** $y_{\downarrow P}$ occurs when the observed guard to the opposing rule becomes (atomically) enabled. We discuss the timing of $y_{\downarrow P}$ in Sections 3.6.3 and Sections 4.2.2.

The **transient fences** for each transition consist of a pair of fixed shapes which are translated so that they begin at the transition-initiation event. We let the **slewtime** $\tau_{y\updownarrow}$ denote the lengths of these transient fences:

Symbol	**Name of Parameter**
$\tau_{y\uparrow}$	$y\uparrow$ slewtime
$\tau_{y\downarrow}$	$y\downarrow$ slewtime

Figure 3.15: Slewtimes of transitions on y.

For each production rule there are two transient fences. The **leading fence** leads the signal and prevents it from transitioning too fast. The **trailing fence** trails the signal, guaranteeing that it transitions within time $\tau_{y\updownarrow}$. This leading/trailing convention, like our circuit equations, makes our arguments for both types of assignments symmetrical with respect to flipping all voltages and currents:

Function	**Name**	**Domain**	**Meaning**
$lead_{y\uparrow}(t)$	$y\uparrow$ leading fence	$t \in [0, \tau_{y\uparrow}]$	$y(t + t_{y\uparrow I}) < lead_{y\uparrow}(t)$
$trail_{y\uparrow}(t)$	$y\uparrow$ trailing fence	$t \in [0, \tau_{y\uparrow}]$	$y(t + t_{y\uparrow I}) > trail_{y\uparrow}(t)$
$lead_{y\downarrow}(t)$	$y\downarrow$ leading fence	$t \in [0, \tau_{y\downarrow}]$	$y(t + t_{y\downarrow I}) > lead_{y\downarrow}(t)$
$trail_{y\downarrow}(t)$	$y\downarrow$ trailing fence	$t \in [0, \tau_{y\downarrow}]$	$y(t + t_{y\downarrow I}) < trail_{y\downarrow}(t)$

Figure 3.16: Transient (finite) partial-fences for transitions on y.

3.6.1 Conditions on DC Fences

1. **Threshold Ordering**. We require that the thresholds have the following ordering:

$$V_{y\downarrow I} < V_{y\uparrow C} < V_{DD} < DC_{y\uparrow} \quad \text{and} \tag{3.5}$$

$$V_{y\uparrow I} > V_{y\downarrow C} > 0 > DC_{y\downarrow}. \tag{3.6}$$

   In Equation 3.5, the $V_{y\downarrow I} < V_{y\uparrow C}$ is necessary so that a signal can remain a logic 1 for an unbounded amount of time. The rest of Equation 3.5 states that V_{DD} is a valid logic 1. Equation 3.6 is the same as Equation 3.5, except that all transitions are negated and all voltages are reflected over $V_{DD}/2$.

   Additionally, for the observation automaton ϕ to be well-defined, we require that

$$V_{y\downarrow C} < V_{y\uparrow C}. \tag{3.7}$$

2. **Drive strength**. Suppose that the inputs[4] to a rule $y\uparrow$ are within the completion regions of logical values that digitally enable $y\uparrow$ (with no assumptions on staticizer state); then any $V \leq V_{y\uparrow C}$ must be a lower fence for $i_y - \eta$. In other words, under worst-case noise the gate can pull the output above the completion threshold when it is completely enabled. There is a similar condition for $y_\downarrow$. These conditions are clearly necessary in order for finite transitions to exist.

3. **Hold strength**. For this constraint view the guard for a production rule $y\downarrow$ in DNF[5]. Suppose that for each disjunct there is some node whose voltage remains below the initiation threshold for that node. Also assume that the intermediate node of the staticizer (if any) is a valid 0. Then $V_{y\uparrow C}$ must be a lower fence for $i_y - \eta$. In other words, under worst-case noise the gate can keep the output above the completion threshold assuming no pulldown disjunct has all of its inputs initiated to 1. There is a similar condition for $y_\uparrow$.

4. **Indefinite DC bounds**. It must be proven that the indefinite DC bounds introduced in Table 3.14 hold for all time.

[4]I.e., terms used in the guard.
[5]Disjunctive Normal Form (DNF), e.g., $(a \wedge b) \vee (a \wedge c)$

3.6.2 Conditions on Transient Fences

Consider each production rule $y\downarrow$. We currently assume the rule's guard has at most two input terms, a and (possibly) b. For this condition we let the inputs be any signals (i.e., any integrable functions of time) in which a and b are initially below their respective initiation thresholds and remain so until arbitrary (but positive) respective times t_a and t_b, when they are bounded by their respective transient fences. Also a and b remain completed after times $t_a + \tau_{a\uparrow}$ and $t_b + \tau_{\uparrow}$, respectively, i.e., the preparation for the next transition is assumed not to occur.

Now consider any output signal $y(t)$ satisfying the circuit model (Inequality 3.3) such that $y(0) > V_{y\downarrow I}$. Let t_y denote the earliest time such that $t_y \leq V_{y\downarrow I}$; then $lead_{y\downarrow}(t+t_y)+\eta$ is an upper fence for i_y and $trail_{y\downarrow}(t+t_y)-\eta$ is a lower fence for i_y.

Similar (but inverted) conditions must also hold for each rule $y\uparrow$.

3.6.3 Slewtime Constraints

Let us again consider the signals a and b used in the preceding section. Without loss of generality (rewriting the guard if necessary), we can also assume that $t_a \leq t_b$. We define the preparation time $t_{y\downarrow P}$ as the as the time at which the guard is atomically enabled:

$$t_{y\downarrow P} \overset{\text{def}}{=} \begin{cases} t_a & \text{if the guard is } a \vee b \\ t_b & \text{if the guard is } a \wedge b. \end{cases} \tag{3.8}$$

We define the **inertial delay** as the delay from preparation to initiation:

Symbol	**Parameter Name**	**Meaning**
$\alpha_{y\uparrow}$	$y\uparrow$ inertial delay	Minimum delay from $t_{y\uparrow P}$ to subsequent $t_{y\uparrow I}$
$\alpha_{y\downarrow}$	$y\downarrow$ inertial delay	Minimum delay from $t_{y\downarrow P}$ to subsequent $t_{y\downarrow I}$

Figure 3.17: Inertial (minimum) delays (α) of transition initiation on y.

We first require that the table shown in Figure 3.17 be valid for all possible signals of the form we are considering. Finally, we require the inertial delays and slewtimes of the circuit to satisfy the following condition:

Definition 7 (Slewtime Constraints) *A circuit satisifes the* **slewtime constraints** *if the following holds for any rules $a\uparrow$ and $b\downarrow$ such that the atomic execution of $a\uparrow$ can (starting in some state) atomically enable some rule $y\downarrow$ while $b\downarrow$ can atomically disable $y\downarrow$. Let $\alpha(a \rightharpoonup b)$ denote the sum of inertial delays along any path through the circuit from a to b. Then the following must hold for any such a, b, and path:*

$$\alpha(a) + \alpha(a \rightharpoonup b) > \tau(a) + \alpha_M(a), \tag{3.9}$$

where $\alpha_M(a)$ is defined either as the minimum $P{-}I$ delay (i.e., $\alpha_M(a) \overset{\text{def}}{=} \alpha(a)$) if the path begins with $y\downarrow$, or as the maximum $P{-}I$ delay if the path does not begin with $y\downarrow$. This condition is necessary for any non-atomic implementation of the atomic PRS model[7].

3.7 The Staticizer Model

As suggested in Figure 3.6, the $i_{stat}(s, y)$ current is the output current of an inverter connected from s to y. To complete the model we must constrain $s(t)$ somehow. We require only that s be a 0 or V_{DD} if y has been completed for long enough:

$$s(t) = \begin{cases} 0 & \text{if } y([t - \tau_{y\uparrow}, t]) \geq V_{y\uparrow C} \\ V_{DD} & \text{if } y([t - \tau_{y\downarrow}, t]) \leq V_{y\downarrow C} \\ \text{arbitrary} & \text{otherwise.} \end{cases} \tag{3.10}$$

Of course, s does not equal 0 or V_{DD} exactly, but noise on s can be translated into current noise using the Lipschitz constant as discussed in Section 3.3.

3.8 The Environment Model

Each node y driven by the environment must satisfy the following conditions:

1. For each initiation event observed at time $t \in t_{y\updownarrow I}$, the signal must be contained in the postulated transient fences (Figure 3.16, Section 3.6). The signal is bounded by the subsequent initiation threshold after each transition.

2. The actual sequence produced by the environment should be coded into additional stable PRs (and possibly nodes[6]) and added to the PRS. The observed events for the new PRs driving y must be (atomically) safe and must obey the minimum delay α_y.

If necessary, the environment may assume that its inputs from the circuit satisfy an infinitesimally weaker (i.e., $\leq\geq$ instead of $<>$) form of condition 1.

Notice that the environment specification is entirely stated in terms of the voltage at y. Therefore the environment must be strong enough to drive the load capacitance at y in a way that meets condition 1.

Because the environment specification directly involves the transient fences at y, these fences should be postulated to have the form given in the circuit's interface specification (simple leading and trailing ramps).

3.9 Summary: The Complete Observation Theorem

Theorem 2 (Observation of an Analog Implementation of a PRS) *Consider any stable, non-interfering PRS together with the unit-capacitance circuit equations of a canonical implementation that has been sized (i.e., all transistor transconductances have been chosen relative to their loads). Suppose furthermore that all disjunctions are mutually-exclusive*[7]*.*

Suppose furthermore that we have found the voltage-level parameters listed below in Figure 3.18, satisfying the DC fence conditions of Section 3.6.1.

Suppose furthermore that we have found the timing and partial-fence parameters listed below in Figures 3.19 and 3.20, satisfying the slewtime and transient fence conditions of Sections 3.6.3 and 3.6.2.

[6]Additional nodes (possibly infinitely many) may be needed to represent the state of the environment. These nodes have atomic transitions. This is a trick so that we can use the extended stability results without modification. The PRs driving these extra nodes exist conceptually but do not have to be supplied in an actual circuit verification.

[7]This is an assumption, also made by others[44], that we make temporarily for simplicity.

Symbol	Name of Parameter
η	noise bound
$V_{y\uparrow I}$	$y_\uparrow$ initiation threshold
$V_{y\downarrow I}$	$y_\downarrow$ initiation threshold
$V_{y\uparrow C}$	$y_\uparrow$ completion threshold
$V_{y\downarrow C}$	$y_\downarrow$ completion threshold
$DC_{y\uparrow}$	DC upper bound on y
$DC_{y\downarrow}$	DC lower bound on y

Figure 3.18: Postulated voltage-level parameters for y.

Symbol	Name of Parameter
$\tau_{y\uparrow}$	$y\uparrow$ slewtime
$\tau_{y\downarrow}$	$y\downarrow$ slewtime
$\alpha_{y\uparrow}$	$y\uparrow$ inertial delay
$\alpha_{y\downarrow}$	$y\downarrow$ inertial delay
$\alpha_{My\uparrow}$	$y\uparrow$ maximum delay
$\alpha_{My\downarrow}$	$y\downarrow$ maximum delay

Figure 3.19: Postulated timing parameters for y.

Function	Name	Domain	Meaning
$lead_{y\uparrow}(t)$	$y\uparrow$ leading fence	$t \in [0, \tau_{y\uparrow}]$	$y(t + t_{y\uparrow I}) < lead_{y\uparrow}(t)$
$trail_{y\uparrow}(t)$	$y\uparrow$ trailing fence	$t \in [0, \tau_{y\uparrow}]$	$y(t + t_{y\uparrow I}) > trail_{y\uparrow}(t)$
$lead_{y\downarrow}(t)$	$y\downarrow$ leading fence	$t \in [0, \tau_{y\downarrow}]$	$y(t + t_{y\downarrow I}) > lead_{y\downarrow}(t)$
$trail_{y\downarrow}(t)$	$y\downarrow$ trailing fence	$t \in [0, \tau_{y\downarrow}]$	$y(t + t_{y\downarrow I}) < trail_{y\downarrow}(t)$

Figure 3.20: Postulated partial-fence parameters for y.

Consider any analog evolution L, i.e., any collection of signals (one per node PRS and one per staticizer intermediate-node) such that

$$|y' - i_y(a, b, s, y)| \leq \eta \qquad (3.11)$$

holds for internal nodes, all $s(t)$ satisfy the staticizer model (Section 3.7), and all external inputs satisfy the environment model (Section 3.8). Suppose that all nodes satisfy the initial conditions (Section 3.4).

Consider the atomic calendar $H \overset{\text{def}}{=} \phi(L)$ consisting of all observed initiation events. H is an execution, i.e., H satisfies (atomic) safety and progress.

Chapter 4

A Proof of the Observation Theorem

We now prove the observation theorem. We assume that we have a SNI PRS and some behavior L satisfying the circuit model and initial conditions. We must prove that $\phi(L)$ is an execution.

4.1 Roadmap

We must show that the events in $\phi(L)$ form a proper execution, i.e., they satisfy safety and progress. Recall that these events are generated by the observation automaton Φ, which acts whenever a signal moves out of its previous threshold-bounded region. Each signal must therefore be shown to be correct by bounding it somehow. Recall (Section 1.6) that we can bound a signal by containing it between locally-conditioned functions called **fences**.

The problem is that we have many signals all depending on one another; the circuit is cyclic. It is therefore not exactly true that there is a multi-dimensional fence that will contain the entire multi-dimensional trajectory L. While it is relatively easy to show that each signal is contained in its fence *assuming all other signals are contained in their respective fences*, the assumption does not hold for a general circuit.

In an acyclic circuit one could use ordinary induction, but our circuits derive much functionality from being cyclic. Therefore, in Section 4.4 I introduce the **Spatial Induction Principle (SIP)** which solves this problem. I prove the principle in Section 4.4.5.

The specific problem with multi-dimensional fences in a *cyclic* circuit is that two or more signals might decide to jump over their fences at *exactly* the same time, with each one "blaming the other for the trouble". To solve this problem, we must hold each logic gate to a slightly higher standard than its input is held: the inputs are bounded by $\leq$ and $\geq$, while the outputs are bounded by $<$ and $>$. Similarly, the inputs will be observed by a weakened observation rule $\overline{\phi}$, while the outputs are observed using the ϕ we have already defined.

The SIP allows us to consider each signal independently. We must use the circuit model to bound each logic-gate's output. Clearly, to get useful bounds we must have bounds on the logic-gate's inputs (just as in static analysis[12]). It is easy to see that the raw bounds that we get from the observation automaton will not be sufficient for this purpose. Therefore, in Section 4.2.5, by extending the observation theorem, we will prove a stronger theorem which implies detailed bounds on all signals in addition to the observation theorem itself.

Finally, to apply the SIP, just as with any induction principle, we must prove the inductive step itself. Here, the inductive step is called a **gate specification**. We prove it in Section 4.4.6.

4.2 The Extended Observation Theorem

To make the observation theorem easier to prove, we make three changes:

1. We infinitesimally loosen the observation rule ϕ, obtaining a new observation rule $\overline{\phi}$ so as to make our gate specifications compatible with the spatial induction principle.
2. We strengthen the result, constructing for each signal y upper and lower bounds u_y and l_y, and proving the **weak boundedness** property:

$$\overline{\textbf{bounded}}(y) \overset{\text{def}}{=} l_y(t) \le y(t) \le u_y(t) \qquad \text{for all } t. \tag{4.1}$$

3. We further strengthen the result, proving that each occurence of each rule r in $\overline{\phi}(L)$ satisfies inertial delay α_r.

4.2.1 The Loose Observation Rule $\overline{\phi}$

Recall from Section 3.5 that ϕ is a projection of the automaton output Φ. As shown in Figure 3.12 (also of Section 3.5), we defined Φ by induction. This definition can be formalized as follows:

$$\Phi(y)_{2i+j} \overset{\text{def}}{=} \text{minimal } t > \Phi(y)_{2i+j-1} \text{ satisfying } \left[(-1)^i y(t) \ge V_{2i+j}\right], \tag{4.2}$$

where $j \in \{0,1\}$ indicates the event type $\{I, C\}$ and V is the repeating sequence of thresholds $\{V_{\uparrow I}, V_{\uparrow C}, -V_{\downarrow I}, -V_{\downarrow C}, \ldots\}$ (of course, omit the first two terms if the initial value of the node is 1).

To prove the spatial induction principle, we will need to allow signals to equal the threhold exactly without triggering the corresponding events. Therefore we change the "$\geq$" to a "$>$":

$$\overline{\Phi}(y)_{2i+j} \overset{\text{def}}{=} \text{infimum } t > \overline{\Phi}(y)_{2i+j-1} \text{ satisfying } \left[(-1)^i y(t) > V_{2i+j}\right]. \tag{4.3}$$

We use the infemum (greatest lower bound) on all t for which the signal has passed the threshold since there will be no earliest such time.

We define lowercase $\overline{\phi}(L)$ as the projection of $\overline{\Phi}(L)$, which selects initiation events. To prove the observation theorem, it suffices to prove that $\overline{\phi}(L)$ is an execution. This is due to the following:

Theorem 3 (Weak Observation) *Suppose it is proved that $\overline{\phi}(L)$ is an execution for any L satisfying noise bound vector $\overline{\eta}$. In other words, $\|y'(t) - i_y(t)\| < \overline{\eta}_y$ for all t, y. Then $\phi(L)$ is an execution for any L satisfying noise bound vector η, provided that $\overline{\eta}_y > \eta_y$ for each signal.*

Proof: consider any L satisfying the tighter noise bound. We construct a new execution L' which has the following repeated modification: for each earliest time at which $\overline{\Phi}(L)_i > \Phi(L)_i$, we add a small bump to L' so that $\overline{\Phi}(L')_i = \Phi(L')_i$. This can clearly be achieved with an arbitrarily short duration, arbitrarily short height bump. Clearly we require the bump to be of sufficiently short duration so it does not affect the next observed event. Furthermore we require the bump to have small enough height so that the loose bound $\overline{\eta}$ is still satisfied and the output current change is also within $\overline{\eta}$. The latter is possible because we have a Lipschitz constant for each gate. L' satisfies the loose noise bound, and $\phi(L) = \phi(L') = \overline{\phi}(L')$, so we conclude that $\phi(L)$ is an execution. □

To apply the result, we use the loose observation theorem, then decrease our noise bounds slightly. We incorporate this reduction into our rounding error bound (and the effect is dwarfed by the true rounding error, of course).

4.2.2 Preparation Events $y\updownarrow P$

Recall that the hold strength condition (Section 3.6.1, condition 3) guarantees that an output remains completed when the opposing guard is not yet enabled. In order to calculate the inertial delay, we will rely on the fact that after the assignment $y\downarrow$ (for example) completes, we have $y < V_{\downarrow C}$, and therefore it takes some time (inertial delay) before y can reach

$V_{\uparrow I}$. Therefore we will use $V_{\downarrow C}$ as a fence after the transition has completed. But at some point, this fence stops holding. We can conservatively assume that it stops holding when the guard g of the next transition (i.e., for the rule $g \rightarrow y \uparrow$) is atomically enabled. We refer to this time as the **preparation event** $t_{\updownarrow P}$:

$$\text{infemum } \left(t \geq (t_{y\downarrow C})_{i-1}\right) \text{ satisfying } g\left(\overline{\phi}(L)(t)\right). \tag{4.4}$$

4.2.3 The PIC Sequence

By the hold strength condition, $y\uparrow$ cannot initiate until some disjunct in the guard has all its terms below their initiation thresholds. If we assume that these terms did not in the recent past (within τ) have preceding transitions, then we can assume that $y\uparrow I$ would not occur until $y\uparrow P$ has first occured:

$$\left(t_{y\uparrow P}\right)_i \leq \left(t_{y\uparrow I}\right)_i. \tag{4.5}$$

Technically, we cannot make the assumption about non-existence of old transitions at this point in the theorem (it would be a circular argument), though in due course we prove that. For now we simply build Inequality 4.5 into our construction by strengthening the infemum domain in Equation 4.4 in the obvious way:

$$\left(t_{y\uparrow P}\right)_i \overset{\text{def}}{=} \text{ infemum } t, \left((t_{y\downarrow C})_{i-1} \leq t \leq (t_{y\uparrow I})_i\right) \text{ satisfying } g\left(\overline{\phi}(L)(t)\right). \tag{4.6}$$

Clearly this satisfies inequality 4.5. Therefore when we insert preparation events into our existing sequence $\overline{\Phi}(L)$, we obtain a sequence of events with the following time ordering:

$$y\uparrow P,\ y\uparrow I,\ y\uparrow C,\ y\downarrow P,\ y\downarrow I,\ y\downarrow C,\ \ldots \tag{4.7}$$

(flip the target-values for a reset value of 1).

4.2.4 Kompletion events $y\updownarrow K$

A consequence of the extended observation theorem will be that

$$(t_{y\uparrow C})_i < (t_{y\uparrow I})_i + \tau_{y\uparrow}. \tag{4.8}$$

We will show this by showing that the transient fences (Section 3.6) are bounds on the rising signal over the interval

$$[(t_{y\uparrow I})_i, (t_{y\uparrow I})_i + \tau_{y\uparrow}]\,, \tag{4.9}$$

and therefore the signal reaches the completion threshold by the end of this interval.

However, as with the $y\updownarrow P$ events, we cannot at this point assume that the right endpoint of this interval occurs before the next event, so we enforce it by using the following right endpoint:

$$(t_{y\uparrow K})_i \overset{\text{def}}{=} \min\Big((t_{y\uparrow I})_i + \tau_{y\uparrow}\ ,\ (t_{y\uparrow P})_{i+1}\Big). \tag{4.10}$$

The corresponding event is called a **kompletion event**.

Finally, we would also like to use $t_{y\uparrow K}$ as an initial left endpoint, so we define:

$$(t_{y\uparrow K})_0 \overset{\text{def}}{=} 0. \tag{4.11}$$

4.2.5 The 3-Phase Decomposition

We refer to the following intervals as **containment phases**:

Phase	**definition**
Prepare$(y\uparrow)_i$	$((t_{y\uparrow P})_i, (t_{y\uparrow I})_i]$
Transition$(y\uparrow)_i$	$((t_{y\uparrow I})_i, (t_{y\uparrow K})_i]$
Hold$(y)_i$	$((t_{y\uparrow K})_i, (t_{y\downarrow P})_{i+1}]$
Prepare$(y\downarrow)_i$	$((t_{y\downarrow P})_i, (t_{y\downarrow I})_i]$
Transition$(y\downarrow)_i$	$((t_{y\downarrow I})_i, (t_{y\downarrow K})_i]$
Hold$(\neg y)_i$	$((t_{y\downarrow K})_i, (t_{y\uparrow P})_{i+1}]$

Figure 4.1: Containment phases.

Notice that this is a partition of time (i.e., of the nonnegative real numbers). In other words, any nonnegative real number is contained in exactly one phase.

4.2.6 Signal Containment Bounds u_y, l_y

We use the 3-phase decomposition described in the preceding subsection to define piecewise **containment bounds** u_y and l_y:

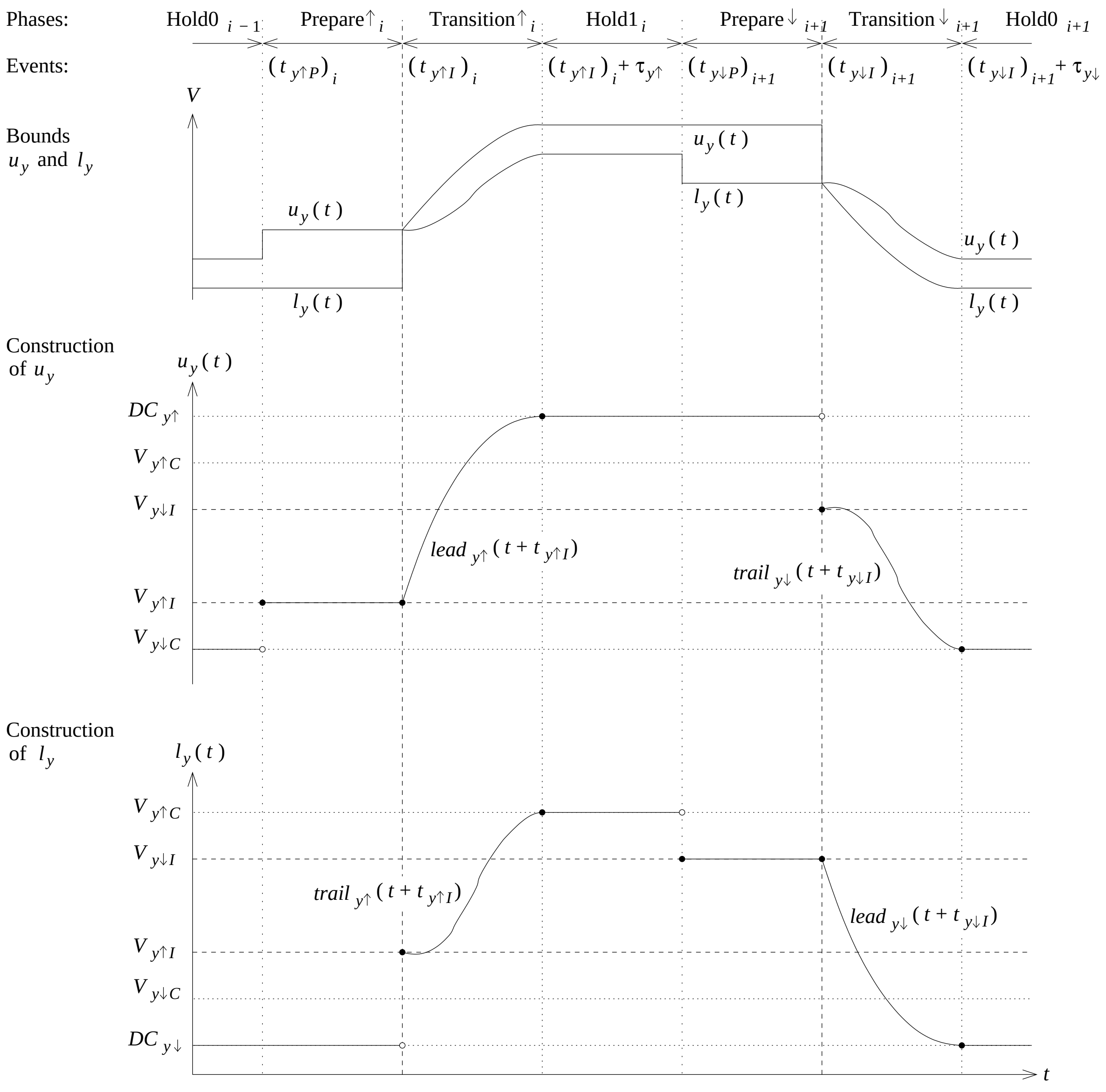


Figure 4.2: 3-phase decomposition, and definitions of u_y and l_y.

Formally, the piecewise functions are defined by the following table:

Phase	$u_y(t)$	$l_y(t)$
Prepare$(y\uparrow)_i$	$V_{y\uparrow I}$	$DC_{y\downarrow}$
Transition$(y\uparrow)_i$	$lead_{t\uparrow}(t+t_{y\uparrow Ii})$	$trail_{t\uparrow}(t+t_{y\uparrow Ii})$
Hold$(y)_i$	$V_{y\uparrow C}$	$DC_{y\uparrow}$
Prepare$(y\downarrow)_i$	$V_{y\downarrow I}$	$DC_{y\uparrow}$
Transition$(y\downarrow)_i$	$trail_{t\downarrow}(t+t_{y\downarrow Ii})$	$lead_{t\downarrow}(t+t_{y\downarrow Ii})$
Hold$(\neg y)_i$	$V_{y\downarrow C}$	$DC_{y\downarrow}$

Figure 4.3: Containment bounds.

4.2.7 Boundedness

We begin by proving that the containment bounds are nonstrict (i.e., $\leq$) bounds on all signals, but we will require the gates to satisfy a slightly stronger specification involving strict bounds. We use the following notation for "bounded up to time t":

$$\textbf{bounded}_t(y) \overset{\text{def}}{=} \Big(s<t \ \wedge \ s\notin\text{Prepare}(y\updownarrow)\Big) \Rightarrow \Big(l_y(s) < y(s) < u_y(s)\Big), \tag{4.12}$$

with which we can express "bounded for all time t" as follows:

$$\begin{aligned} \textbf{bounded}(y) &\overset{\text{def}}{=} \Big(t\notin\text{Prepare}(y\updownarrow)\Big) \Rightarrow \Big(l_y(t) < y(t) < u_y(t)\Big) \\ &= \textbf{bounded}_t(y) \text{ for all } t. \end{aligned} \tag{4.13}$$

We cannot require strong boundedness during the Prepare phases, since equality holds at the right endpoint (and potentially other points, since we are using $\overline{\phi}$), but weak boundedness holds by construction during these phases.

Finally, we can speak of the entire execution L being bounded:

$$\begin{aligned} \textbf{bounded}(L) &\overset{\text{def}}{=} \textbf{bounded}(y) \quad \text{for all signals } y \\ \overline{\textbf{bounded}}(L) &\overset{\text{def}}{=} \overline{\textbf{bounded}}(y) \quad \text{for all signals } y. \end{aligned} \tag{4.14}$$

4.2.8 α-Safety (Inertial-Delay Safety)

In the extended observation theorem we will prove that every signal y satisfies an inertial delay α_y on all its transitions. Formally, we define:

$$\begin{aligned} \alpha\text{−}\mathbf{safe}_t(y) &\overset{\text{def}}{=} \Big(s{<}t \ \wedge \ s \in [t_{y\updownarrow I} - \alpha_y, t_{y\updownarrow I}]\Big) \Rightarrow g(\phi(L(s))) \\ \overline{\alpha\text{−}\mathbf{safe}}_t(y) &\overset{\text{def}}{=} \Big(s{<}t \ \wedge \ s \in [t_{y\updownarrow I} - \alpha_y, t_{y\updownarrow I}]\Big) \Rightarrow g(\overline{\phi}(L(s))). \end{aligned} \tag{4.15}$$

Similarly, we define the non-subscripted $\alpha\text{−}\mathbf{safe}(y)$, $\alpha\text{−}\mathbf{safe}(L)$, and $\overline{\alpha\text{−}\mathbf{safe}}(\ldots)$ as we did for $\mathbf{bounded}(\ldots)$ and $\mathbf{bounded}_t(\ldots)$.

4.3 The Progress Argument

Suppose we have proven all of the extended observation theorem, except for the progress portion, i.e., suppose we have shown that

$$\mathbf{bounded}(L) \ \wedge \ \alpha\text{−}\mathbf{safe}(L). \tag{4.16}$$

We claim that progress must be satisfied.

In the following proof we assume the PRS contains no unstable disjuncts. While this condition is not necessary, it simplifies the proof.

Suppose progress is not satisfied. Consider some rule $y\downarrow$ which is enabled indefinitely in $\overline{\phi}(L)$. We can assume that some disjunct is enabled indefinitely. By the form of l_y, we know that each term a in this disjunct must eventually remain above $V_{a\uparrow C}$. By the drive strength condition (Section 3.6.1, condition 2) the output must decrease under these conditions until it has passed all the way down to $V_{y\downarrow C}$. On the way there it must cross $V_{y\downarrow I}$ and generate an event. This contradicts the assumption that $y\downarrow$ was indefinitely enabled. ▫

4.4 The Spatial Induction Principle (SIP)

4.4.1 Input Hypotheses

For any PR of the form

$$g(x_1, \ldots, x_k) \rightarrow y\uparrow \tag{4.17}$$

we refer to the following conditions as the **input hypotheses**:

i. $\overline{\textbf{bounded}}(x_1, \ldots, x_k)$, i.e., $\overline{\textbf{bounded}}(x_i)$ for all i.

ii. In $\overline{\phi}(x_1, \ldots, x_k)$, the atomic guard for $y\uparrow$ does not have two transitions within $\tau_{y\uparrow} + \alpha_{My\uparrow}$ of one another (refer to Section 3.6.3 for the definition of $\alpha_{My\uparrow}$).

iii. In $\overline{\phi}(x_1, \ldots, x_k)$, the atomic guard for $y\uparrow$ does not have two transitions within ϵ of one another even when some x_i are delayed by up to $\max(\tau_{y\uparrow}, \tau_{x_i\uparrow})$.

iv. In $\overline{\phi}(x_1, \ldots, x_k)$, atomic noninterference $(\neg g \vee \neg q)$ always holds, even when some x_i are delayed by up to $\max(\tau_{y\uparrow}, \tau_{x_i\uparrow})$.

As usual, similar conditions for $y\downarrow$ are made by mirroring all target values.

4.4.2 Output Hypotheses

For any node y we refer to the following conditions as **output hypotheses**:

I. $\textbf{bounded}(y)$

II. $\alpha-\textbf{safe}(y)$

4.4.3 The Statement of the SIP

For every node y, recall that the (logic) **gate** is the pair of PRs whose target node is y. We therefore refer to the conjunction of the input hypotheses for $y\uparrow$ and $y\downarrow$ as the input hypotheses for y. The gate is correctly implemented if these hypotheses imply that the output hypotheses hold:

Definition 8 (Gate Specification) *Suppose we have arbitrary signals (i.e., arbitrary continuous functions of time) $x_1, \ldots, x_k$ not necessarily satisfying any circuit model, but satisfying the input hypotheses. Suppose furthermore we have a signal y satisfying the initial conditions and circuit model for the gate y. The* **gate specification** *for gate y is the predicate that if the preceding assumptions hold, then the output hypotheses hold for y.*

The **Spatial Induction Principle (SIP)** simply states that if all gate specifications hold, then all output hypotheses hold:

Theorem 4 (SIP) *Suppose all gate specifications hold for a PRS implementation. Consider any behavior L satisfying the initial conditions and circuit model. Then all output hypotheses hold for L.*

I prove the safety portion of the observation theorem in two steps. In Section 4.4.5, I prove the spatial induction principle itself. In Section 4.4.6, I apply the spatial induction principle, by proving that all gate specifications hold.

4.4.4 Extended Stability

Theorem 5 (Extended Stability Application) *If $\alpha-\mathbf{safe}(L)$ holds, then input hypotheses ii-iv hold for all nodes.*

This is purely a statement about atomic executions of PRS, which I have previously shown[7], and which is the subject of Chapter 5.

4.4.5 A Proof of the SIP

Clearly if all output hypotheses were satisfied then all input hypotheses would be satisfied, for $\mathbf{bounded}(L) \quad \Rightarrow \quad \overline{\mathbf{bounded}}(L)$ by definition, and extended stability gives us the atomic input hypotheses. The difficulty is that a priori we do not have all input hypotheses satisfied; we only have each individual hypothesis satisfied under the assumption that all others hold. The trick is to show that if some input hypothesis is not satisfied, then there is a particular gate we can blame. This is in analogy to the standard proof of ordinary induction (if the hypothesis did not hold for some integer, then we blame the hypothesis of the earliest integer for which it did not hold).

As we have just seen, it suffices to prove

$$\mathbf{bounded}(L) \ \wedge \ \alpha-\mathbf{safe}(L). \tag{4.18}$$

Consider any particular L satisfying the initial conditions and circuit model. For a contradiction, assume the negation of Equation 4.18. Let t_{Err} be the latest time such that

$$t < t_{Err} \quad \Rightarrow \quad \mathbf{bounded}_t(L) \ \wedge \ \alpha-\mathbf{safe}_t(L). \tag{4.19}$$

There must be some "bad" signal y for which t_{Err} is indeed a least upper bound: the above consequence is false at y for any $t > t_{Err}$. In other words:

$$\neg\mathbf{bounded}_{t_{Err}}(y) \;\vee\; \neg\alpha\mathbf{-safe}_{t_{Err}}(y). \tag{4.20}$$

We now construct a modified "evolution" L^1 in which $y^1(t)$ satisfies the circuit model, but other signals do not necessarily behave physically. We will reach a contradiction by demonstrating that $x_1^1, \ldots, x_k^1$ satisfy the input hypotheses, while y^1 does not satisfy the output hypotheses.

Let $\overline{\phi}_{<t_{Err}}(L)$ denote the prefix of $\overline{\phi}(L)$ consisting of all events occuring before time t_{Err}. For each x_i, let $u_{x_i}^1$ and $l_{x_i}^1$ denote the signal containment bounds constructed in the standard way from $\overline{\phi}_{<t_{Err}}(L)$. Choose some continuous function $x_i^1(t)$ satisfying the following:

$$\begin{cases} x_i^1(t) = x_i(t) & \text{for all } t \leq t_{Err} \\ l^1(t) \leq x_i^1(t) \leq u^1(t) & \text{for all } t > t_{Err} \end{cases}. \tag{4.21}$$

Notice that the weak inequalities are essential because the inputs could have reached the containment bounds at time t_{Err}. This is why we use the weak $\overline{\mathbf{bounded}}(x_1, \ldots, x_k)$ in our input hypotheses.

Now integrate these functions to obtain a signal $y^1(t)$ satisfying the following:

$$\begin{cases} y^1(t) = y(t) & \text{for all } t \leq t_{Err} \\ |i_y(t, x_1^1, \ldots, x_k^1) - \mathrm{d}y^1(t)/\mathrm{d}t| \leq \eta & \text{for all } t > t_{Err} \end{cases}. \tag{4.22}$$

Equation 4.20 is a safety violation, and hence does not depend on the execution after time t_{Err}, giving us:

$$\neg\mathbf{bounded}_{t_{Err}}(y^1) \;\vee\; \neg\alpha\mathbf{-safe}_{t_{Err}}(y^1). \tag{4.23}$$

This violates the gate specification for y because $x_1^1, \ldots, x_k^1$ satisfy the input hypotheses.

□

4.4.6 The Application of the SIP

Recall (Section 4.4.3, Theorem 4) that the spatial induction principle can be applied whenever all gate specifications hold. We now proceed to show that the gate specification of each gate y holds.

Suppose we have signals $x_1, \ldots, x_k$ and y as assumed by the gate specification (Section 4.4.3, Definition 8). The input signals $x_1, \ldots, x_k$ satisfy the input hypotheses, and the output signal y satisfies the circuit model. We must now show the output hypotheses, i.e.:

$$\mathbf{bounded}(y) \wedge \alpha{-}\mathbf{safe}(y) \qquad \text{(must be shown).} \tag{4.24}$$

4.4.6.1 Boundedness

We now show $\mathbf{bounded}(y)$. Recall (Equation 4.14) that $\mathbf{bounded}(y)$ is a condition holding for all t:

$$\mathbf{bounded}(y) \overset{\text{def}}{=} \Big(t \notin \text{Prepare}(y\updownarrow)\Big) \Rightarrow \Big(l_y(s) < y(s) < u_y(s)\Big). \tag{4.25}$$

We prove that the above inequality holds strongly during the Transition and Hold phases (recall Figure 4.1) and weakly during Preparation phases. Our proof is by induction on the phases, so that at the beginning of each phase we can assume that the signal was at least weakly bounded by the bounds of the previous phase (whenever the containment bounds u and l are themselves continuous). The first phase (starting at $t = 0$) is always a Hold_0 phase, and the left endpoint bound for this phase is the initial condition on y.

4.4.6.2 Boundedness: $\mathbf{Hold}(y)_i$

Suppose $t \in ((t_{y\uparrow K})_i, (t_{y\uparrow P})_{i+1}]$. At the left endpoint the signal is completed, i.e.,

$$V_{y\uparrow C} \leq y((t_{y\uparrow K})_i) \leq DC_{y\uparrow} \tag{4.26}$$

because this condition held at the end of the preceding phase, assuming no containment-bound discontinuity. By construction of $t_{y\uparrow K}$ (Section 4.2.4), a containment-bound discontinuity could only occur in this case if the atomic guard for $y\downarrow$ were enabled within $[(t_{y\uparrow P})_i, (t_{y\uparrow P})_i + \tau_{y\uparrow})$. But this violates a combination of input hypotheses: by noninterference (iv) it implies that the guard of $y\uparrow$ has two transitions within $\tau_{y\uparrow}$ of one another, violating (ii).

We must now show that the condition of Inequality 4.26 continues to hold (in strengthened form) until $(t_{y\uparrow P})_{i+1}$. This amounts to the hold strength condition (Section 3.6.1, Condition 3) which requires that we show that in each disjunct for the rule for $y\downarrow$, some term is below its initiation threshold.

Suppose some disjunct has all its terms at voltages above its initiation threshold. By construction of the Hold phase, the guard for $y{\downarrow}$ is not atomically enabled, so one of these terms has an atomic value of 0 over the entire interval. Therefore, by (i), the only way that this term could have a high voltage is if it recently (up to time $\tau_{x_i\downarrow}$ ago) had a downgoing transition. This contradicts (iii): we could perturb this transition to create a pulse in the guard.

Finally, the Hold Strength condition makes an assumption about the staticizer, which follows from the staticizer model (Section 3.7).

4.4.6.3 Boundedness: Prepare$(y{\downarrow})_i$

Suppose $t \in ((t_{y\uparrow P})_i, (t_{y\uparrow I})_i]$. We must show that $DC_{y\downarrow} < y(t) \leq V_{y\uparrow I}$. The indefinite DC bound holds as a condition to the theorem (Section 3.6.1, Condition 4). By inductive assumption, the bound holds at the left endpoint, and no seams ever occur in the containment bounds there. The right endpoint is, by construction of $\overline{\phi}$, the earliest point beyond which this inequality stops holding. □

4.4.6.4 α-Safety and Transition$(y{\downarrow})_i$ Boundedness

The observation theorem assumes that the inertial delay (Section 3.6.3) and transient fence conditions (Section 3.6.2) hold for $y{\downarrow}$ in a controlled setting where each input has exactly one upgoing transition during the Prepare$(y{\downarrow})$ and Transition$(y{\downarrow})$ phases, and the output is initially above $V_{y\uparrow C}$. Our inductive boundedness proof gives us the latter; the former remains to be shown.

For now (as in Sections 3.6.2-Section 3.6.3) we assume a 1- or 2-input gate with inputs a and (possibly) b. First, suppose the guard is $a \wedge b$ (NAND, C-element, or INV). If there are two transitions within the back-to-back Prepare and Transition phases, they must occur within time $\tau_{y\uparrow} + \alpha_{My\uparrow}$ (the maximum length of the two phases) of one another. By assumption, the guard holds at the beginning of the Prepare phase, so it must be invalidated soon thereafter, in contradiction of input hypothesis (ii).

For a guard $a \vee b$, multiple transitions also invalidate the guard because we assume mutual exclusion for the inputs of NOR gates. This assumption holds for the examples we are studying, but is not technically necessary. FenceCalc allows multiple transitions on the later-arriving input. Technically, to take advantage of this, we would have to allow this in the theorem condition as well.

4.4.6.5 Environment-Driven Nodes

If y is driven by the environment in response to $x_1, \ldots, x_k$, then it satisfies boundedness and safety constraints (Section 3.8) which imply that the gate specification holds for y.

Chapter 5

Extended Stability

Recall (Section 2.2.2) that **stability** (for a nonvacuously enabled rule) requires a guard to continue to hold until the target is assigned. This requirement was designed to avoid **glitches** (undesired pulses). Electrically, it ensures that the driving circuit does not cut off before the target has reached its final value.

Unfortunately, as we have seen, a signal's digital value is not in general a direct instantaneous function of the value of the analog signal. Therefore it is possible that a driving circuit is enabled when the digital guard is not. Clearly this condition can only exist temporarily, immediately after a transition. Therefore there is concern when multiple transitions arrive at a gate in rapid succession.

For example, suppose that two opposing transitions into a NAND gate arrive too close to each other. An output dip can occur that was not atomically predicted, as shown below:

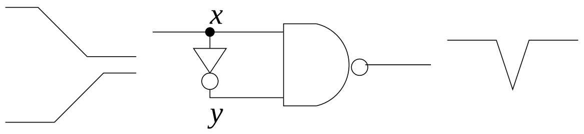


Figure 5.1: Two opposing transitions arriving at different gate inputs in rapid succession.

This failure can be viewed as an *atomic* instability that occurs when the inputs are perturbed (i.e., shifted slightly in time with respect to one another). This principle (of digital instability under perturbation) allows us to rule out such problems by defining a new notion of **extended stability** over the domain of atomic executions.

Extended stability states that all guards should be stable, *even under slight perturbations of the inputs.* It is purely a property of atomic executions. However, it is clearly also timing-dependent: we can only guarantee such a property if we have a notion of inertial delay in our atomic model. We therefore begin by defining this notion in Section 5.1.

In Section 5.2 we proceed to define how inertial delay adds (abstractly) over paths. In Section 5.3 we prove the **minimal perturbation** principle, which extends event perturbations to entire executions. In Section 5.4 we show how these minimal perturbations are related to path delays. Finally, in Section 5.5 we prove the **extended stability theorem**s, which (under certain **path-delay constraints**) guarantee that inputs do not arrive too close (in a precise sense).

5.1 Minimum-Delay Annotations

We begin by expanding the atomic timestamp model to include minimum-delay annotations with semantics. A priori, the absolute values of timestamps are meaningless, as all timestamps can be scaled without changing the safety or progess of a trace. However, as noted in Section 2.4.4, 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 9 (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** *(i.e., is α-***safe***) if and only if each time at which a rule r becomes enabled and the next earliest time at which r is executed is always at least $\alpha(r)$. Said another way, whenever the rule executes at time t, it was enabled over the time interval $(t - \alpha(r), t]$.*

We choose to define delay-annotation semantics in the atomic model, since such a semantics naturally extends to the non-atomic model: an execution has valid delay timing in the non-atomic model if and only if its observation has valid delay timing in the atomic model. This gives the expected non-atomic semantics because we chose our observation rule (Section 3.5) so that the atomic timestamp would be equal to the time at which propagation was enabled in the non-atomic model.

5.2 Propagation Paths

We now describe feedback delays in terms of the annotations developed in the previous section. 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 10 (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 11 (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.

5.3 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: perturbation of any (atomic timestamp) execution by delaying an event e_1 *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 5.4), 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 $\Delta \overset{\text{def}}{=} (E, \delta)$ is defined as follows:

Definition 12 (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 13 (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. □

5.4 The 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 6 (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 $e_0 \overset{\text{def}}{=} e$. 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 $e_\Delta \overset{\text{def}}{=} e_k \in E$ and $r(e_k), \ldots, r(e_0)$ a path. The base case holds as e_0 is delayed and forms a trivial path. For the inductive step we assume some e_i is delayed. 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 $t(e_0) - t(e_k)$. □

5.5 The Extended Stability Theorem

5.5.1 Path-Delay Constraints

Assume the PRS comes with some annotation τ[1] which associates a positive real number $\tau(r)$ to each rule r. This number will constrain the minimum delay on any path between any two rules on which r has the opposite types of dependence:

Definition 14 (Safety Constraint) *A PRS* **satisfies the path-delay constraints** *if it has delay and slewtime annotations such that for all rules r depending positively on r' but negatively on r'', and p a path between r' and r'' (in either direction), we have:*

$$\tau(r) < \alpha(p). \tag{5.1}$$

[1]Throughout this thesis we use τ to represent slewtime because slewtime must satisfy path-delay constraints for our main result. Of course, in the atomic model slewtime has no meaning. For an abstract result we therefore use the τ symbol here only to bound path delays.

5.5.2 The Theorem Statement

The theorem assumes we are given all of the following:

1. An (atomically) stable PRS with delay annotations α.
2. Any (atomic) execution H such that $\alpha-\mathbf{safe}(H)$.
3. Path Delay annotations τ satisfying the path delay constraints.
4. Any rule r of the form $g(x_1, \ldots, x_k) \ \rightarrow \ y := v$.

The consequence of the theorem has two parts:

I. g never has two transitions within $\tau(r)$ of one another if $y \neq v$ when the first transition occurred.

II. Suppose $y \neq v$ over some open interval containing some time t. For a decreasing sequence of ϵ consider a calendar H'_ϵ where delays less than $\tau(t)$ – i.e., delays in some perturbation Δ_ϵ – are carried out but no other inputs are changed. Then, for some (sufficiently small) fixed ϵ', the guard value g does not have two transitions both within ϵ' of t.

Intuitively, part I says that g has no pulses shorter than $\tau(r)$, and part II says that we cannot cause arbitrarily short pulses in g by perturbing its inputs in any way (moving any events we want by up to τ).

5.5.3 A Proof of Part I

Suppose for a contradiction that part I is violated: g had two transitions within $\tau(r)$ of one another, and $y \neq v$ when the first transition occurred. By stability, the first transition was $g \uparrow$, and the order of all transitions (on g and r) was $g\uparrow \ ; \ r \ ; \ g\downarrow$.

Let $G\downarrow$ be a set of events – on which r negatively depends – which occured after the $g\uparrow$ and which suffice to cause $g\downarrow$. Consider the minimal (r,τ)-perturbation. Some event $e \in G\downarrow$ was delayed, for else $H_{(r,\tau)}$ is an unstable execution containing $g\uparrow; g\downarrow; r$.

The event e follows r by less than τ. Therefore, by the propagation property, there is some propagation path p, from r to e, with $\alpha(p) < \tau(r)$. This violates a path delay constraint. ▫

5.5.4 A Proof of Part II

Suppose for a contradiction that part II is violated: there is some time t contained in an open interval T for which $y \neq v$. Furthermore, for any ϵ there is some perturbation Δ_ϵ, less than τ, leading to a calendar H'_ϵ in which those perturbations are carried out but no other inputs are changed.

Now consider the *execution* H_{Δ_ϵ}, which is the minimal Δ_ϵ-perturbation. Since the PRS is stable, we can assume that over T the guard value $g(H_{\Delta_\epsilon})$ has at most one transition.

Therefore, there is an interval T_ϵ over which the guards disagree, i.e., $g(H'_\epsilon) \neq g(H_{\Delta_\epsilon})$ over the entire interval, and the interval size decreases with ϵ. The interval size must decrease in the sense that its endpoints converge to each other, and the endpoints coincide with events $begin(T_\epsilon)$ and $end(T_\epsilon)$ that change guard g. Clearly, the guard depends positively on one of these events and negatively on the other. Therefore the guard's type of dependence on one of these events differs from its type of dependence on some event in Δ_ϵ which preceded it by less than time $\tau(r)$. The path delay constraint on the path between these two events is violated. ▫

5.6 Conclusion

In this chapter we have shown that when a simple timing model is added to the atomic timestamp model for PRS, transitions cannot arrive at a gate's input in rapid succession so as to cause glitches. Specifically, in part I of the theorem we saw that each individual input has a minimum time between transitions. In part II we saw that if we perturb the arrival times of different inputs within a small time window, we do not find a glitch in the guard. A "small time" is defined in terms of the propagation delays through the circuit.

Extended stability is a self-contained result about atomic PRS executions. However, it is also a central component of our proof of the observation theorem, used in Section 4.4.4. Ultimately, in Section 4.4.6 we use extended stability to assume that each input to our gate has a single isolated transition during the transition phase of the output.

Chapter 6

FenceCalc™

FenceCalc™ – a 6,000-line Modula-3 program I have written – facilitates circuit verification by checking that the conditions of the observation theorem that we discussed in Chapter 3 hold for any given circuit. Given any circuit, if such assumptions exist, they can be found (in time linear in the circuit size) by the program using a novel form of Breadth-First Search (BFS) discussed in Section 6.7.3.

FenceCalc™ postulates the theorem conditions by associating tentative DC and transient hypotheses to each transition. The hypotheses are locally `update`d until they satisfy containment conditions. At each step, FenceCalc™ reports a **containment distance** which allows us to detect divergence or convergence of the overall algorithm. When the containment distance is small enough, the theorem conditions are met by the current set of hypotheses.

When used with a standard script (see Appendix E.1), the program terminates successfully only if such fences are found and all conditions needed for rigorous circuit verification have been checked.

6.1 The Input/Output Specification

The circuit to be verified is given in `KAST` form. `KAST` is a restricted form of `CAST`[55] with extensions that specify the relative transconductance of each transistor and the current noise of each node.

Recall that the observation theorem assumes that a number of quantities have been postulated for each node y: the current-function i_y and noise-bound η_y (Section 3.3), I/C thresholds (Section 3.5), and the partial fences $DC_{y\updownarrow}$, $lead_{y\updownarrow}$, and $trail_{y\updownarrow}$ (Section 3.6). These quantities are handled by FenceCalc™ as indicated below:

sym.	**theorem param.**	**source/destination**	**representation**
i_y	current-function	`KAST` input file	`FcNode.T`
η_y	current-noise	`KAST` input file	`FcNode.T`
$V_{y\updownarrow I}$	initiation thresholds	`KAVL` input (manual or defaulted)	`FcEpsilon.T`
$V_{y\updownarrow C}$	completion thresholds	`KAVL` inp. (automatically calculated)	`FcEpsilon.T`
$DC_{y\updownarrow}$	DC bounds	postulated; checked and/or output	`FcHypothesis.DC`
$lead_{y\updownarrow}$	leading transient fence	postulated; checked and/or output	`FcHypothesis.AC`
$trail_{y\updownarrow}$	trailing transient fence	postulated; checked and/or output	`FcHypothesis.AC`

Figure 6.1: FenceCalc™ I/O quantities associated with each node y.

6.2 User's Guide

FenceCalc™ implements a number of operations that are sufficient to verify a circuit. The functions are generally applied in the following sequence:

1. Input circuit equations and initiation (I) thresholds from user.
2. Find global DC bounds using synchronizing BFS.
3. Compute completion (C) thresholds.
4. Compute all AC (transition) hypotheses using synchronizing BFS.
5. Check the constant hypotheses (i.e., check consistency of user-provided thresholds).
6. Check all AC hypotheses.
7. Check slewtime constraints.
8. Subtract calculation error from η, yielding **effective eta**[1]. The effective eta must be positive, and is the bound on noise allowed in the rigorous result.

These operations are accessible through the `KAVL` language (see Appendix E) and can be performed by a universal `KAVL` script given in Section E.1.

If all the checks are successful, then the assumptions of the observation theorem are satisfied. At this point the results of the program are conceptually plugged into a template proof which assumes the facts checked in steps 5-8 and proves circuit correctness. In practice we do not use an elementary proof checker, so this last step is purely conceptual: we are done when the above operations complete successfully. Thus, to achieve provably *correct construction* in the VLSI design methodology, it suffices to run the above FenceCalc™ operations successfully as the last step of circuit synthesis.

[1]The command is actually called `eta-eff`.

6.3 Circuit Representation

A circuit is represented as a set of nodes; each node is uniquely labeled (`name`) and comes with additional structure as shown below:

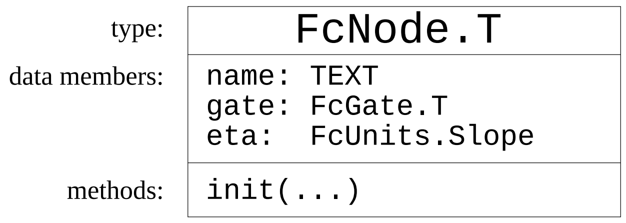


Figure 6.2: The `FcNode.T` data structure.

The `gate` substructure lists the fanin nodes and defines the logic gate connected to those nodes. `eta` is a floating-point number (in units of current, or `FcUnits.Slope`) representing the target noise bound.

6.3.1 The Transition Representation, `FcTransition.T`

Recall that a target assignment consists of a target node and a target value. The `FcTransition.T` data structure represents any such assignment:

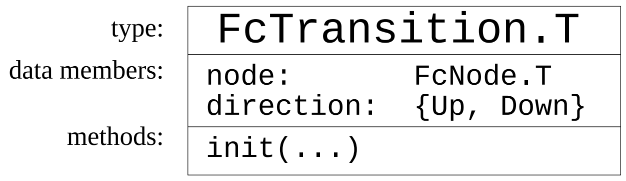


Figure 6.3: The `FcTransition.T` data structure.

Each hypothesis in FenceCalc™ is associated to an `FcTransition.T` instance.

6.3.2 The System Representation, `FcSystem.T`

FenceCalc™uses a large data structure, `FcSystem.T`, to represent the entire circuit. This data structure allows retrieval of the sets of all nodes and transitions, and provides generic graph abstractions for both, allowing a single BFS implementation to be used on either graph.

The `FcSystem.T` instance represents only the circuit, not the hypotheses that are being postulated for that circuit. That way, multiple sets of hypotheses could be postulated for a single circuit (in principle).

6.4 The Transition Hypotheses, `FcHypothesis.T`

At every step of a verification algorithm in FenceCalc™, the currently postulated DC and AC (i.e., transient) hypotheses are represented for each transition and – by extension – for the whole system.

6.4.1 Data Representation

Each transition hypothesis consists of an initiation threshold and a completion threshold (represented together using `FcEpsilonPair.T`), a DC bounding value, and, additionally, for an AC hypothesis, the leading and trailing fences.

Despite these differences, both types of hypotheses inherit a common abstraction, the `FcHypothesis.T`, which allows the hypothesis to be `update`d based on the present input hypotheses (retrieved from a `FcSysHypo.T` – see the next section) or `output` for monitoring. This common superclass allows a single BFS (or checking) algorithm to be used directly on both types of hypotheses. The complete class hierarchy is as follows:

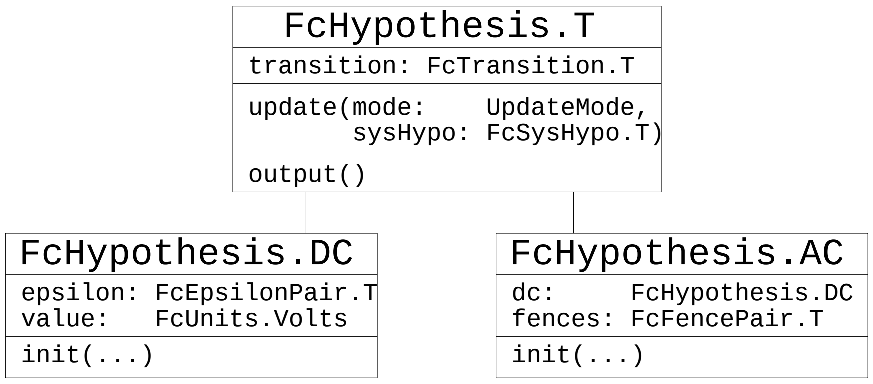


Figure 6.4: The `FcHypothesis.T` type hierarchy.

The `UpdateMode` enumeration is defined in Modula-3 as follows:

```
TYPE
  UpdateMode = {Check,      (* leave all hypotheses unchanged      *)
                Accumulate, (* accumulate onto existing hypotheses *)
                Replace     (* replace existing hypotheses         *)
  };
```

This control parameter determines whether the updating `Replace`s existing hypotheses for initial estimation, `Accumulate`s in a conservative way, or is for a final `Check`.

6.4.2 Abstract Operation of the `update` Method

A call to `update` can only be unsuccessful if the mode is `Check`. In other modes, `update` changes the hypothesis on the output, using the calculations described in the previous section.

A successful call to `update` on a transition on y guarantees that the hypothesis for that transition satisfies the conditions of the theorem, but the conditions may be destroyed for the next stage (i.e., any gate which has y as an input).

Each call to `update` returns a **containment distance** which indicates how close the hypothesis was from being satisfied at the beginning of the call. When `update` is used in `Check` mode on every hypothesis in the system and the containment distance is everywhere small enough to be analyzed as rounding error, the verification is complete. As we proceed with many passes of `update`s (as described in Section 6.7.3), we can use the sequence of containment distance to determine if the overall algorithm is converging or diverging.

6.4.3 Transient Fence Data Representation

Recall from Section 3.6 that a transient fence for transition $g \to y\uparrow$ consists of a pair of functions, *lead* and *trail*, defined over $[0, \tau_{y\uparrow}]$. Furthermore, (Section 3.6.2) we require these functions (when appropriately translated) to bound the output, assuming a single transition on each input appearing in g. As noted in Section 1.6, a general way to achieve this is by requiring *lead* and *trail* to be mathematical fences (satisfying the local conditions discussed in Appendix C).

In general, a fence is not tight: it is not in general true that under extreme noise the evolution comes arbitrarily close to the fence. However, our goal is to have tight results, i.e., we would like our bounds to be actually achieved in the worst case. Therefore we specifically compute **isodynamic fence**s, defined as follows:

Definition 15 (Isodynamic Fence) *An* **isodynamic fence** $y(t)$ *– of the upper variety – for* $i_y(t, y)$ *satisfies* $y'(t) = i_y(t, y(t)) + \eta$ *for some constant* η.

A lower isodynamic fence is of course similarly defined with $-\eta$. I chose the word "Isodynamic," meaning "equal strength," because these fences have the same strength η everywhere. In such a fence, there is no single "weak link" that constrains η.

In FenceCalc™, `hyp(`y`).update(`$\ldots$`)` computes isodynamic fences by integrating $i_y + \eta$ and $i_y - \eta$ using Euler's method[8]. Thus in practice the fences are not exactly isodynamic, but can be arbitrarily close. We discuss the Euler error in Section D.5. The final output (using the `eta-eff` command of Appendix E.2.4) takes all error sources into account, giving a rigorous result.

To achieve uniform accuracy, Euler's method evaluates a gate's input values regularly every `stepsize` seconds and produces a piecewise-linear output, represented by a new data point at the same time as each evaluated input point. All hypotheses are piecewise-linear functions represented by one data point for each `stepsize` seconds.

To avoid interpolation and aliasing problems, a standard `stepsize` is used for all calculations. This allows us to represent all functions as discrete lists of points. Furthermore, we synchronize our input and output calculations[2] so that evaluating the inputs is simply a matter of selecting a point from the input lists.

6.5 The `FcScenario.T` and Input-Cube for Multi-Input Gates

In general, a logic gate has many inputs. A transient hypothesis for the output must hold, assuming each input has at most one transition (Section 3.6.2). These transitions can happen at arbitrary times with respect to one another. However, for simplicity, we do not explore the multidimensional space of relative arrival times using Charlie Diagrams[41]. Instead, we assume that one of the inputs is always critical, as we now illustrate.

Consider a 2-input NAND gate, with inputs a and b, initially 0. Suppose that a rises first, long before b does, at time t. Notice that the pulldown is cut off, so we can ignore the transitions on a. In fact, we can ignore the analog value of a completely without affecting the calculation. In general, for a conjunctive rule, we can – correctly and with minimal loss of generality – replace the leading fence of the earlier transition by a vacuous DC bound as shown below:

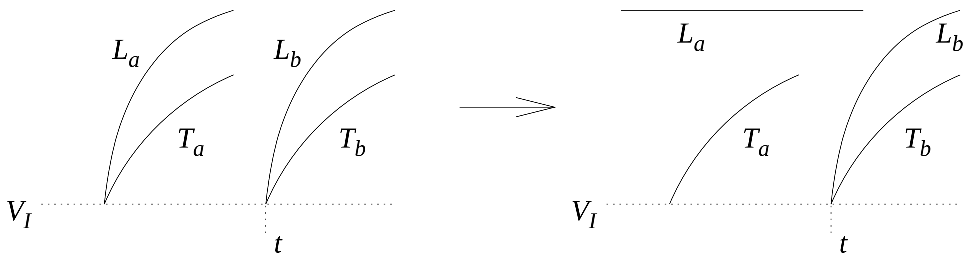


Figure 6.5: Replacing the ignored leading bound L_a by a DC bound.

Furthermore, if we assume that the trailing fence is monotonic (a condition easily checked at the end of the computation), then the trailing bound on any input transition on a is also

[2]This is sometimes overconservative, but by an amount which decreases with `stepsize`.

described (more weakly) as the same trailing bound, delayed to time t, as shown below:

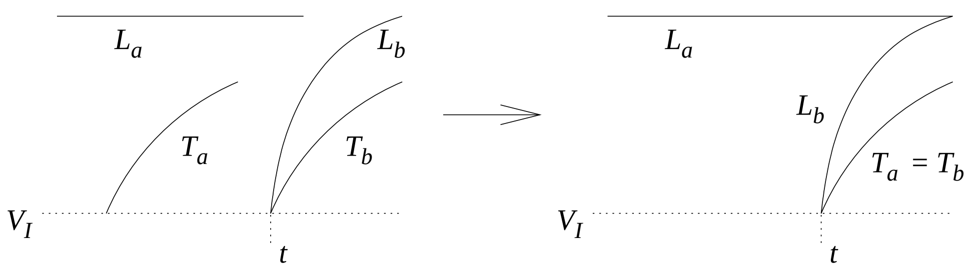


Figure 6.6: Shifting T_a to time t, without loss of generality.

This step involves no loss of generality (i.e., these are the tightest possible bounds we can give on the input) because the case where both inputs cross the threshold at the same time does indeed occur.

The above transformations allow us to express all input bounds on a single (1-dimensional) time scale, assuming we have chosen a particular scenario (`FcScenario.T`). The scenario identifies a critical input; thus the number of scenarios is at most equal to the number of transistors in the pulldown network.

In summary, we can choose a single time t before which the guard is not yet initiated. For each timestep after time t, given a scenario and a set of input hypotheses, we can retrieve a single **input cube**[3] which associates a voltage interval to each input.

6.6 The Algorithm For the Transient `update` Method

As discussed in the previous section, for each scenario we obtain a sequence of input cubes that bound all inputs in lock-step. We use the input cubes to compute the **partial hypothesis** for each scenario, using the following local condition: at each timestep, we ensure that our output slope bounds the worst-case output current over all possible input values in the input cube. We discuss the details of this calculation in Section D.3.

Finally, once we have a set of partial hypotheses, we combine them conservatively to obtain an output hypothesis which satisfies the theorem conditions over all possible scenarios.

[3]Our input cube is a set of *analog* voltage ranges, not to be confused with digital "input cubes" one encounters when working with Karnaugh maps.

6.6.1 Synchronized Calculation of Leading and Trailing Bounds

Recall that a transient output hypothesis must bound the output transition after the signal has crossed $V_{yI\uparrow}$. However, this time is not exactly known. In fact, we can only bound the output initiation time (relative to the critical input initiation time) using the bounds $\alpha_{y\uparrow}$ and $\alpha_{My\uparrow}$, which we must compute. In addition to computing $\alpha_{y\uparrow}$ and $\alpha_{My\uparrow}$, we must ensure that our output fence holds, given any actual inertial delay in $[\alpha_{y\uparrow}, \alpha_{M\uparrow}]$.

First, we compute $\alpha_{y\uparrow}$ and $\alpha_{My\uparrow}$. Since the output is known to be in $[V_{y\downarrow C}, DC_{y\downarrow}]$ at time t, we can extend both endpoints to isodynamic fences until they both reach $V_{y\uparrow I}$. Then $t + \alpha_{y\uparrow}$ and $t + \alpha_{M\uparrow}$ are, respectively, the resulting times of intersection with $V_{y\uparrow I}$.

Recall that our output bounds are defined relative to the *actual* time t_o that the output first crossed $V_{y\uparrow I}$. However, since this time is unknown, we begin by computing a *trailing* bound assuming $t_o = t + \alpha_{y\uparrow}$.

Here we combine the assumption that the output-current function is monotonic in the inputs (Section 3.2.4) with the assumption that the fences are monotonic (previous section) to argue that the trailing bound we just computed is valid for any *actual* t_o. In general, $t_o \geq t + \alpha_{y\uparrow}$, and the input fences are assumed monotonic, so in general the input voltages will be equal or higher to the ones we used. Therefore the output currents will be equal or higher to the ones we used, and our trailing bound holds for any actual t_o.

Similarly, we compute a *leading* bound assuming $t_o = t + \alpha_{My\uparrow}$.

The preceding calculation can be coded using exactly two sequences of Euler's Method output bounds, `LeadTrail` and `TrailLead`, that operate in parallel on a common sequence of input cubes. Initially, $ModeLT$ is "leading bound" and $ModeTL$ is "trailing bound," and the timestep number i is 0. The inner loop consists of the following sequence of steps:

- Retrieve input cube C for the next timestep, number i after critical input
- Compute Euler step `LeadTrail` using C with bounding mode $ModeLT$
- Compute Euler step `TrailLead` using C with bounding mode $ModeTL$
- If `LeadTrail` has crossed V_I, then
  - Let $ModeLT$:= "trailing bound"
  - output $\alpha = i * \texttt{timestep}$
- If `TrailLead` has crossed V_I, then
  - Let $ModeTL$:= "leading bound"
  - output $\alpha_M = i * \texttt{timestep}$

Notice that each input cube is used in exactly two calculations: the `LeadTrail` calculation, which is initially a leading bound but becomes trailing, and the `TrailLead` calculation, which is initially trailing but becomes leading.

6.7 The System Hypotheses, `FcSysHypo.T`

The system hypothesis represents all currently-postulated parameters required by the Observation Theorem. Like the transition hypotheses, the system hypothesis has an update algorithm which brings it closer to satisfying the conditions of the theorem. A correct update algorithm results in satsifying all of the conditions simultaneously and therefore yields a rigorous verification. We discuss correct system-hypothesis–updating algorithms in Sections 6.7.2-6.7.4.

6.7.1 Data Representation

At present I am using verification algorithms that (at every step) require exactly one hypothesis of each type (AC/DC) for each possible transition. Therefore, for each type of hypothesis I use a data structure, `FcSysHypo.T`, which associates a hypothesis to each transition using the standard Modula-3 hashmap generic, instantiated as `FcTransHypoTbl.T`:

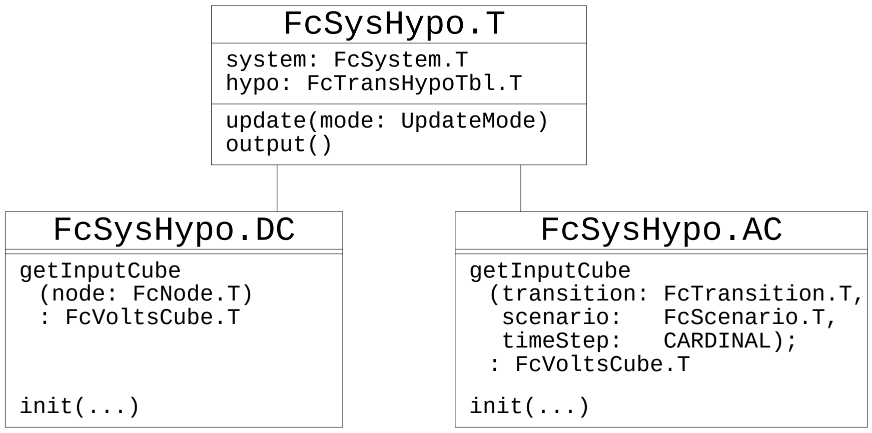

FcSysHypo.T
system: FcSystem.T
hypo: FcTransHypoTbl.T
update(mode: UpdateMode)
output()
FcSysHypo.DC
getInputCube (node: FcNode.T) : FcVoltsCube.T
init(...)
FcSysHypo.AC
getInputCube (transition: FcTransition.T, scenario: FcScenario.T, timeStep: CARDINAL); : FcVoltsCube.T
init(...)

Figure 6.7: The `FcSysHypo.T` type hierarchy.

Given such a mapping, I define the `getInputCube` operation, which returns an `FcVoltsCube.T`, given any gate's output node. The `FcVoltsCube.T` associates each of the gate's inputs to the interval of possible voltages that are assumed in the partial-fence's condition.

For an AC hypothesis this cube varies as time passes after the latest or earliest input (depending on the gate type). Thus the form of `FcSysHypo.AC.getInputCube` must indicate the specific `scenario` and `timeStep` that the cube is computed for. As with transition hypotheses, however, there is a common abstraction for `update`ing and `output`ting the hypotheses for the entire system. In this case, the `update`ing is a synchronizing BFS algorithm, which we discuss next.

6.7.2 The Nondeterministic Update Algorithm

For now, suppose every PR is exercised at least once. We now discuss algorithms which find valid hypotheses for verification of an analog implementation, if they exist. We will discuss both a simple algorithm (in this section) and a faster, more sophisticated algorithm (in Section 6.7.3).

We begin with the simple algorithm. Suppose that some hypotheses H satisfying the theorem conditions exist. The following algorithm knows the I/C thresholds but does not know H initially. It discovers some hypotheses $H' \subseteq H$ [4] satisfying the theorem conditions:

Definition 16 (Nondeterministic Verification Algorithm) *The* **nondeterministic verification algorithm** *consists of the following two phases:*

1. **Transition capture**. *Simulate the system until each node has transitioned twice (according to the observation rule). Define* $lead_{y\updownarrow} = trail_{y\updownarrow}$ *and let this be the portion of the signal from* $[(t_{y\updownarrow I})_1, (t_{y\updownarrow C})_1]$*, shifted to begin at* $t = 0$*. Define the DC bounds as the most extreme values ever observed in simulation.*

2. **Fence bloating**. *Choose some small tolerance* δ*. While some fence condition is violated by more than* δ*, fairly choose a random transition and* `update` *its hypothesis using the* `Accumulate` *mode.*

We now prove the correctness of this algorithm[5]. Let the ghost variable H' denote the current set of transition hypotheses in the algorithm. First, we show that it suffices to prove the invariant $H' \subseteq H$. Since updates are done in `Accumulate` mode, they form the chain $H'_0 \subseteq H'_1 \subseteq \cdots \subseteq H$. Since the $\subseteq$ correspond to inequalities and H is a compact space, the sequence converges. Because the sequence converges, the changes to the hypotheses become ever smaller and converge to zero. With probability 1, each transition is selected infinitely often. Therefore each time a particular transition is selected, a smaller change is required to make that transition satisfy the hypotheses. Therefore the **containment distance**, or amount by which the hypotheses are not satisfied, goes to zero, and the algorithm terminates. When the algorithm terminates, the fence conditions are violated by at most δ, which can be taken into account by `eta-eff`.

[4]By $H' \subseteq H$, we mean that if any set of complete signals L satisfies $\mathbf{bounded}(L)_{H'}$ then it also satsifies $\mathbf{bounded}(L)_H$. Clearly this can be expressed as a set of inequalities directly on the fences.

[5]As this is a probabilistic algorithm, we show that the algorithm terminates with probability 1, and that it only terminates with a correct result.

Now we just have to prove the invariant $H' \subseteq H$. Clearly $H'_0 \subseteq H$ because the simulation is bound by the theorem since we assumed H exists and satisfies the theorem assumptions. Suppose the invariant were violated by some fence. Since `update` calculates the fences by solving the circuit equation with noise $\pm\eta$, there is a simulation L with the corresponding worst-case noise which equals that fence. Therefore we would have $\neg\textbf{bounded}(L)_H$, a contradiction. ▫

6.7.3 The Synchronizing BFS Algorithm

The preceding algorithm is correct, but the fence-bloating phase is unnecessarily slow. Besides the obvious runtime uncertainty associated with randomized algorithms, the algorithm performs redundant computations to reach some global valid hypothesis assignment H.

Consider the transient hypotheses, which are propagated over the transition graph (i.e., henceforth a "node" in this graph abstraction is a actually a `FcTransition.T`). The key observation is that once we have viable hypotheses (i.e., hypotheses that equal those in H) for all external input nodes and all nodes in some transition-graph cutset, then each remaining hypothesis can be computed in constant time using the following algorithm:

Definition 17 (Synchronizing BFS Algorithm) *The* **synchronizing BFS algorithm** *assumes that some set of nodes S_0 has been* `update`*d initially. Between steps of the algorithm, let S denote the set of nodes that have been* `update`*d. The algorithm performs the following step repeatedly: some node not in S whose inputs have* **all** *been updated is updated. When there is no such node left, the algorithm terminates.*

This differs from the standard (non-synchronizing) BFS algorithm by requiring **all** inputs of each new node to have been updated, not just one of them. FenceCalc™ implements this algorithm for both the `FcTransition.T` graph (for transient and DC hypotheses) and the `FcNode.T` graph (for DC hypotheses).

Thus we should focus our efforts on getting the hypotheses for the cutset to converge as fast as possible (any other computation would be wasted CPU cycles). Thus, each `update` should use the newest data available. To accomplish this, we simply perform a number of synchronizing BFS **passes**. Since each new node is updated only when **all** of its inputs were updated in the same pass, no node must be `update`d twice.

Each pass requires one `update` for each node in the circuit. Because all operations are locally linear (in the neighborhood of H), the number of passes required is proportional to the constant $\log(\delta)$, where δ is the exit tolerance (a.k.a. `bfs-tol`). Therefore (for fixed δ

and fixed fence representation complexity) the algorithm runs in time linear in the size of the circuit.

6.7.4 Eliminating the Transition Capture Step

As explained in Section 6.7.2, the objective of the transition capture step was to initially establish the invariant $H' \subseteq H$ over all nodes. We now discuss how this can be done without looking at a simulation of the system.

The transition-capture phase of the algorithm is potentially tedious because it involves simulating the circuit (with real data over many cycles, potentially) until each node transitions twice. We propose two ways to eliminate this step, the **bottom-symbol** method and the **ramp-initialization** method.

If the circuit has no internal initially-enabled PRs, we can do away with the transition-capture phase of the algorithm using **bottom symbol**s. We simply begin with the fences for the external inputs (which are part of the environment specification), and all other hypotheses are assigned the **bottom** symbol. We only update a transition if none of its inputs are **bottom**. This maintains the invariant that $H' \subseteq H$ over all nodes that are not **bottom**. Since circuit transition is exercised, all **bottom**s disappear eventually, and the algorithm is correct by the arguments of Section 6.7.2. However, this method has the drawback that to avoid deadlock, we must define `update(...)` for hypotheses on multi-input gates that have one or more input hypotheses **bottom**.

In the **ramp-initialization** method, each non-environment transition hypothesis is initialized to a simple piecewise-linear ramp shape. After a few stages of propagation, the hypotheses converge to fixed-slewtime shapes, by the arguments given in Appendix B.2. From that point on, the fence-bloating step proceeds as normal. In practice, this method works as well as the preceeding methods.

6.8 Conclusion

Using FenceCalc$^{\text{TM}}$ with a standard script (see Appendix E.1), we can find and check the conditions of the observation theorem for any circuit when those conditions exist.

We will use FenceCalc$^{\text{TM}}$ in two ways. First, we will use it to guide the selection of thresholds and, possibly, delay elements, which we need in order to verify the circuit. As we will see in Chapter 7, we can ensure that the observation theorem conditions exist by properly choosing observation thresholds (Section 7.1) and, as a last resort, adding delay to the circuit (Section 7.6). Since it is always possible to guarantee the existence of the theorem conditions, and FenceCalc™ finds such conditions if they exist, we conclude that we can always use FenceCalc™ (and the methods we will discuss in the next chapter) to find theorem conditions.

Finally, we use FenceCalc™ to check the conditions of the observation theorem so that we can conceptually apply the theorem to our circuit. When substituted into the theorem, the conditions produced by FenceCalc™ give us a complete rigorous verification: the 4-threshold observation of any possible analog evolution is an atomic execution.

Chapter 7

Rigorous Verification Results

I have successfully produced rigorous verifications of two non-trivial circuits using FenceCalc™. For both circuits it is assumed that the relative transconductance (see Section 3.2.1) of NMOS transistors is twice that of PMOS transistors, that staticizers have 5% of the strength of other gates, and that all transistors have a threshold V_T that is 20% of V_{DD}.

I have rigorously verified both circuits assuming that a current-noise of any form whose peaks are as large as 0.1% of the maximum drive current of any gate can appear on that gate's output at any time. This is after taking error in the calculation itself into account. To put this 0.1% in perspective, notice that we are using a pessimistic absolute noise measuring method: the currents are measured relative to the maximum current that could theoretically occur. Clearly we cannot hope for such a bound to be more than 5% (or even 2%, given our p/n ratio) as this would obviously lead to examples where the staticizer is overpowered.

7.1 The Choice of Observation Thresholds

In all circuits I have verified, there was an arbitrary choice of initiation thresholds $V{\updownarrow}I$. This was the only arbitrary choice, because the methods of Section 6.7.3 guarantee that if fence parameters corresponding to a given $V{\updownarrow}I$ choice exist, then FenceCalc™finds them.

To satisfy the hold strength condition (Section 3.6.1), we must have

$$V_{y\uparrow I} < v_0 < V_{y\downarrow I}, \tag{7.1}$$

where v_0 is the logic threshold. Furthermore, if the above condition (adjusted slightly for noise) holds, then we can always verify the circuit by adding more delay to the circuit.

The circuits I have verified have 5- and 7-stage feedback loops, meaning that they run at 10 and 14 transitions per cycle, respectively. This is as fast as any reasonable circuit runs[22]. I have rigorously verified them using FenceCalc™, without adding any additional delay beyond the delay that brings them to this cycle time.

I selected the initiation thresholds by first arbitrarily setting them to the transistor v_T thresholds. This worked for a trivial ring oscillator but not for the synchronized loops or WCHB chain. I raised the initiation thresholds on every node in which a slewtime constraint was not met or on which the *lead* and *trail* fences diverged earliest. For completely automatic verification, a generic search should be used.

7.2 A Synchronized Pair of Loops

The first non-trivial circuit I have verified is a synchronized pair of loops. This circuit has internally enabled PRs and no environment (it is closed). As noted above, the relative transconductance of NMOS transistors is twice that of PMOS transistors. I have chosen an NMOS relative transconductance of 1 (volt/second)/(volt^2). [1]

The circuit is successfully verified with all nodes having effective $\eta = 0.20$ volts/second, which corresponds to 0.5% of the maximum drive slewrate of any gate. This is after taking error in the calculation itself into account. Before calculation error, $\eta = 0.35$. The circuit, η, V_{DD}, and the transistor parameters are given in `KAST` form, which we obtain by labeling all nodes in the circuit, as follows:

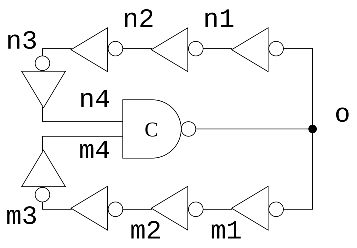


Figure 7.1: Synchronized pair of inverter rings.

See Section F.1 for the details of the verification calculations for this circuit.

[1]To apply the result to a circuit of a particular scale, the "second" can be replaced by some other time unit (e.g., FO4s, or 13.7ps) without affecting the result. Therefore the absolute unit of time is irrelevant for our purposes, and we use the "second," which is easiest to write.

7.3 A Chain of Dual-Rail WCHB Buffers

parameter:	v_{DD}	v_T	k_n	k_p	staticizers
value:	5V	1V	2 $s^{-1}V^{-1}$	1 $s^{-1}V^{-1}$	5%

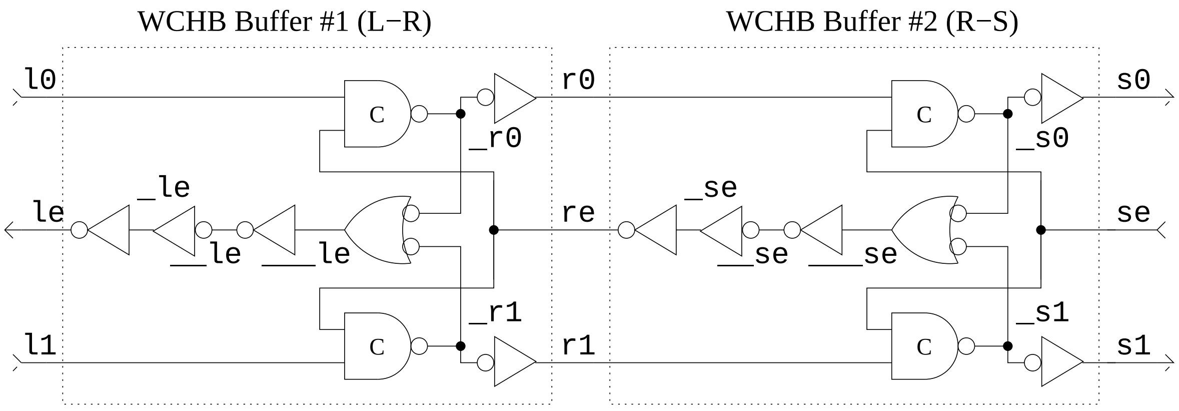


Figure 7.2: Chain of dual-rail WCHB buffers.

As with the synchronizing loop circuit, we must specify the I/C thresholds that define the observation rule. In addition, however, the WCHB chain has an environment, so we must specify the hypotheses on the environment's outputs using the `tran` command.

The circuit is successfully verified with all nodes having effective $\eta = 0.05$ volts/second, which corresponds to 0.1% of the maximum drive slewrate of any gate. This is after taking error in the calculation itself into account. Before calculation error, $\eta = 0.09$. In the next two subsections, we will discuss how this was achieved.

7.3.1 Example: WCHB Verification, with Delay Insertion

To begin with, we set all I (initiation) thresholds to 1V, and we let FenceCalc™ infer the tightest possible completion thresholds from this choice. We run FenceCalc™, and BFS diverges – the original circuit cannot be modified with these thresholds. However, as we discuss in Section 7.6, any circuit can be verified[2] if we add restoring delay elements to internal cycles. In the case of our WCHB chain, the node Re is in all internal cycles, so it suffices to add delay elements at Re. This result is verifiable in FenceCalc™:

User Action	**FenceCalc™ Response**	η
Set all I thresholds to 1V (relative to 0 or V_{DD})	BFS diverges	any
Insert 6 inverters at Re	Success	$\leq .03$ V/s

Figure 7.3: Delay Insertion for WCHB Verification.

7.3.2 Example: WCHB Verification, with Careful Threshold Selection

While the delay-insertion method is very general, it is obviously preferable to avoid modifying (and possibly slowing down) the circuit that we are attempting to verify. It is sometimes possible to do this by selecting the observation thresholds more carefully than we did in the previous subsection. I have verified the WCHB chain shown in Figure 7.2 without adding additional delay (beyond what is already in the figure), as follows:

User Action	**FenceCalc™ Response**	η
Set all I thresholds to 1 (relative to 0 or V_{DD})	BFS diverges	any
Raise all I thresholds to 1.6	$I > C$ violated at C-element	any
Adjust I at C-elem inputs back to 1	Success	≤ 0.03
Fine-tune the thresholds	Success	≤ 0.09
Account for arithmetic error	Success	$\leq \mathbf{0.05}$

Figure 7.4: Careful Threshold Selection for WCHB Verification.

[2]assuming steady-state correctness

The final effective of $\eta = 0.05$ is the rigorous value claimed at the beginning of this section. See Section F.2 for complete details of the verification calculations for this circuit.

7.3.3 Noise-Sensitivity Calculation

For a given circuit, FenceCalc™ gives us a "pass" or "fail" for any postulated noise bound η. By the results of Chapter 6, we know that there is some critical η which separates the "pass"es from the "fail"s. Thus we have a continuous metric that allows standard search algorithms to be employed for noise-sensitivity optimization and threshold selection.

Furthermore, we can explore η as a multi-dimensional space, consisting of an independent noise-bound for every node in the circuit. The following experiment illustrates such an exploration.

For each node x, we will compute a noise tolerance. Let η_0 denote the largest noise bound which gives a successful verification when all nodes are allowed noise up to η_0. The noise tolerance η_x is defined as the largest noise bound we can have on x while all other nodes have noise bound $\eta_0/2$. For the WCHB chain, I have estimated η_x/η_0 using FenceCalc™, as shown below:

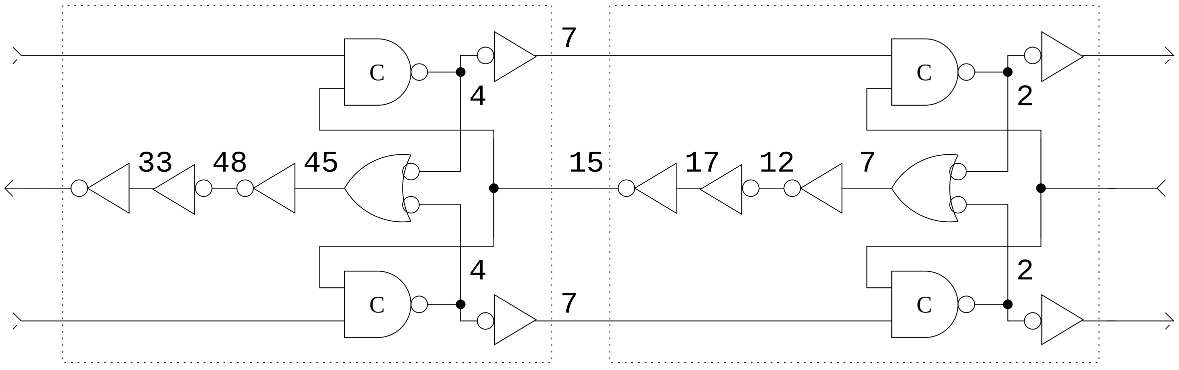


Figure 7.5: Noise Tolerance of Individual Internal Nodes of the WCHB Chain.

Lower numbers indicate more sensitivity to noise; larger numbers indicate more tolerance. In this result, the left handshake has an unrealistically large noise tolerance, because there is no model for slewtime transfer through the environment, i.e., the environment generates a fixed transition shape regardless of how bad the transitions *to* the environment are.

7.4 Fence Resolution Requirements

The preceding circuits can be verified with just 100 points per transient fence if we ignore Euler error. However, for a rigorous result we need 2K-20K points per fence, depending on which Euler error estimation method is used (Section D.5).

The small stepsize is needed primarily for the Euler error estimation. Therefore it is not necessary to use it when first postulating the hypotheses. For example, suppose 8 BFS phases are required (3-8 was typical). We could use a large stepsize for the first 6 phases and a smaller stepsize for the last 2, for a speedup factor of 4. This would require upsampling the hypotheses in between phases. This speedup was not used for the results in this thesis.

7.5 Parameter-Space Searches

In finite time, we can use our verification method to prove that a circuit is correct over any closed region of parameter space. Traditional methods would require infinite time. A closed region can be covered by a constant number of smaller closed patches. In each of these patches we model the parameter uncertainty as dynamic noise and rigorously verify the circuit assuming this extra dynamic noise is present.

This technique requires a finite amount of computation, but, unfortunately, the amount of computation is proportional to the number of patches, and hence inversely proportional to the average patch size. The patch size is limited by the dynamic noise we can handle.

7.6 Generality of the Method

If we add enough delay to the circuit, we can rigorously verify any implementation of PRS, provided the circuit and implementation technology satisfy certain restrictions, as follows:

1. We assume that the technology satisfies the conditions given in Section 3.2.4. It can be directly seen that these conditions hold for CMOS. One of those conditions (slewtime restoration) implies that a particular ring of identical inverters can be verified in FenceCalc™.[3] Suppose that in that verification some inverter's transient hypothesis (i.e., the set of transient fences for both types of output transitions) is S. Suppose additionally that in that verification, on reset, the ring oscillator was initialized with a step function.

2. The actual **noise on the delay elements** that we add is strictly less than that used to claim slewtime restoration for the technology.

3. **Steady-state correctness** of the implementation. Thus, there exist $V_{\updownarrow I}$ and $V_{\updownarrow C}$ thresholds such that the DC conditions of the theorem are satisfied everywhere. This assumption is clearly necessary; without this assumption, there would be no logically-valid voltages. We are free to choose any thresholds (with the proper ordering between I and C) that have logically-valid voltages. For simplicity, in this section we assume fixed threholds that are independent of the circuit node (this adds no restriction if the gates are fully steady-state–composable in any combination).

We are given that some ring oscillator of some length l is verified with a transient hypothesis S. Therefore, after $2l$ inverters, S is mapped back to S. The ring oscillator was verified with some noise which we assumed was greater than the noise on the actual delay elements that we will add. Therefore, we can assume that S is mapped back to rS (i.e., S compressed in time by a factor of $1/r$) for some $r < 1$.

Suppose that we have a circuit which failed verification. Cut all cycles, and postulate S at all newly dangling inputs. By assumption, the dangling outputs do not all map to S (for then we would have a complete verification), but they map to some finite-width parallelogram B.

A very slow delay element (i.e., a pair of inverters with very low relative transconductance in both stages) has an inertial delay equal to the width of B and therefore treats

[3]Slewtime restoration must actually be checked over a range of inverter parameters; I have checked it for CMOS in FenceCalc™, assuming we can set $V_T = 0.2V_{DD}$; this could be extended to a large region using the method described in Section 7.5.

B as a step function. By assumption, a step function maps to S in k steps for some k. Therefore, after k delay elements, B is mapped to r^nS, for some n. After a total of $k+n$ delay elements, B is mapped to S. Therefore the verification is complete at that point.

The process is summarized in the following figure:

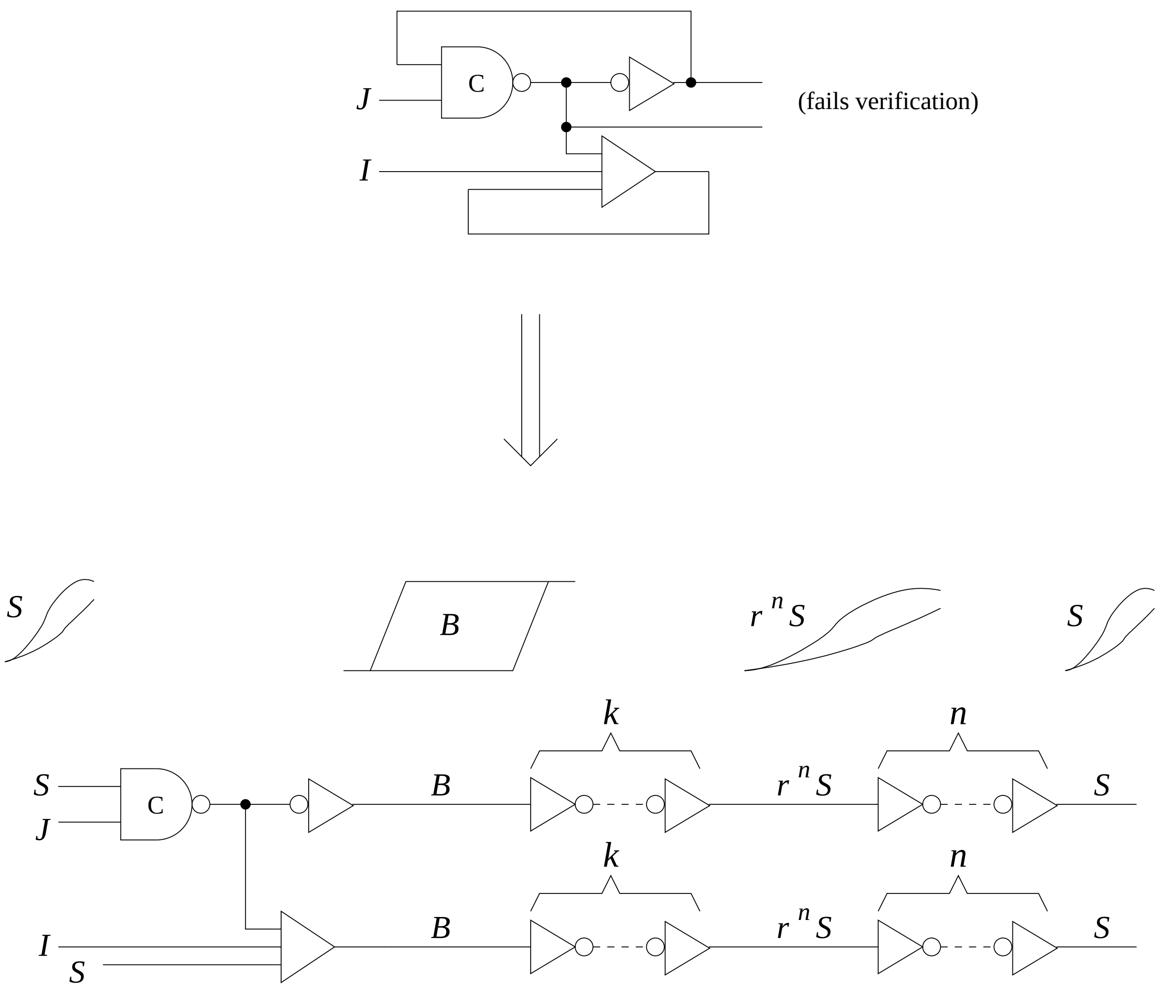


Figure 7.6: Adding delay elements to a circuit to make it pass verification.

Of course, this argument only serves to show that there is an upper bound on the number of delay elements required, assuming we do not change the I/C thresholds from their default values. The actual number required will be smaller, and we will only add delay after first attempting to choose the thresholds properly. In the circuits I have rigorously verified, I added at most two inverters beyond the theoretical minimum (which is 3 inverters on each loop, for CMOS).

7.7 Limitations of Single-Bound Hypotheses

FenceCalc™ produces a tight calculation in the sense that if the theorem conditions can be met, then FenceCalc™ will find valid conditions. However, if the theorem conditions cannot be met and the procedure described in the preceding section is not used to correct the problem, then as BFS proceeds in FenceCalc™, the hypotheses diverge as shown below:

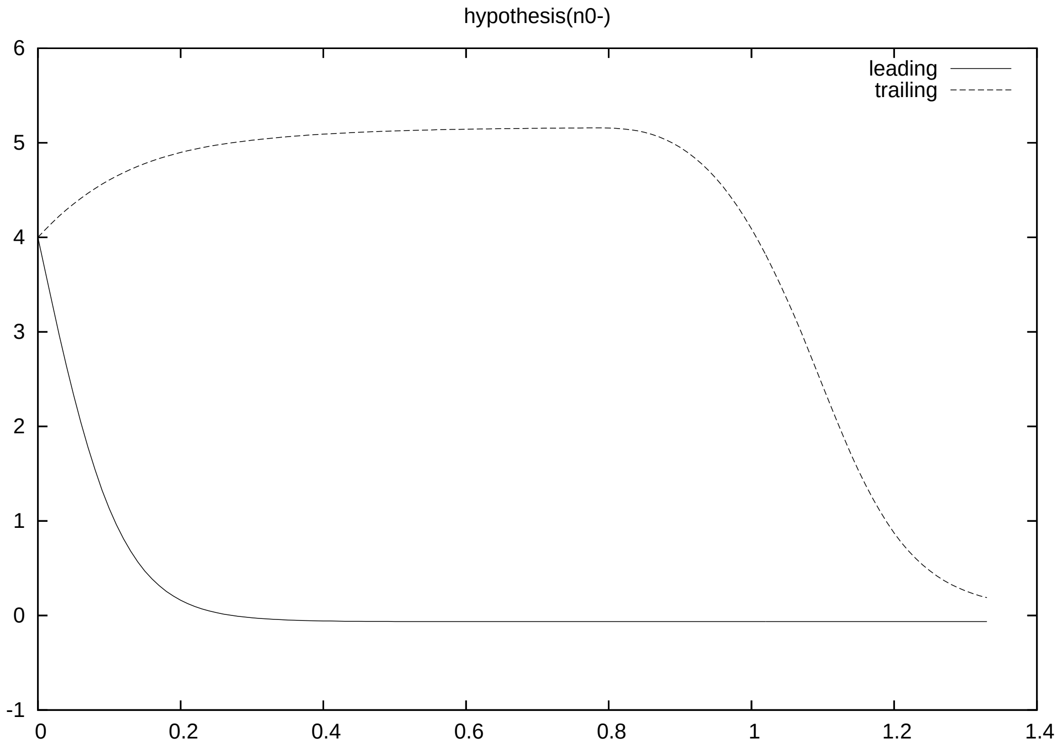


Figure 7.7: Diverging hypotheses: If the leading and trailing fences are too far apart, then they continue to drift away from each other.

As we have seen in Section 6.7.2, this implies that conditions satisfying the theorem cannot be found. However, this does not necessarily mean that the unmodified circuit is unverifiable.

In our present theorem formulation, there is some slop in our transient fence conditions (Section 3.6.2). Specifically, we assume that the input can be *any function* bounded by the transient hypothesis. This clearly allows many possibilities that do not actually occur in the analog model.

For example, suppose that the hypothesis of an input to an inverter contains an extremely wide rectangle, such as is the case in Figure 7.7. Below we sketch the function contained in the hypothesis bounds which was used to compute the worst-case trailing fence.

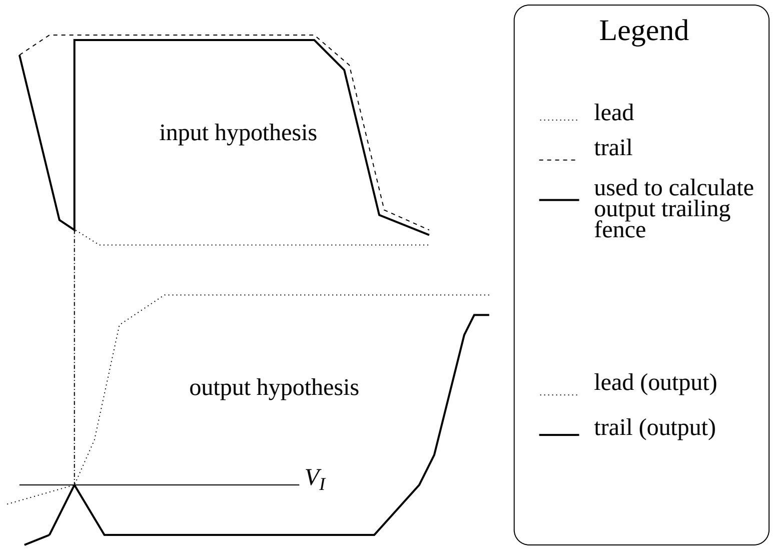


Figure 7.8: Function within input-hypothesis bounds that is used to calculate the trailing output-hypothesis bound.

We can sometimes solve this problem by using multiple sets of hypotheses correpesponding to different scenarios. For example, consider our verification of the synchronized pair of loops (Section 7.2). The PRS for this circuit contains initially-enabled internal PRs. Therefore, as discussed in Section 3.4.1, we must assume that at reset the node with the initial transition is driven with the transient hypothesis for that node.

In fact, we can also verify that circuit assuming the initial transition is a step function, but not using our present version of the observation theorem. Consider the following three possibilities for the hypotheses of the node containing the initial transition:

1. The step function (from the reset circuitry).

2. The steady-state hypothesis computed by BFS.

3. The conservative (but tight) combination of 1 and 2 expressed in the same way: as a *single* pair of fences.

We know the circuit is verifiable using hypothesis (2). However, this does not allow hypothesis (1), which we would like to allow (for the reset transition). Nonetheless, FenceCalc$^{\text{TM}}$ reports that if we use hypothesis (1), then, after a few stages of propagation, the results are contained in (2).

Unfortunately, a calculation using (3) fails entirely, for the reasons discussed above. Nonetheless, we have seen that we can temporarily have multiple hypotheses for a single transition corresponding to different scenarios, and merge them later when they agree.

Chapter 8

Conclusions

My primary intellectual result is the demonstration of the soundness of PRS and its operational model in terms of elementary mathematical concepts. It has been argued that a system designer should not deal in operational models. This is a valuable principle, but there always comes a time when we ask whether the model that the designer has used really matches the underlying operational model of the machine. The underlying operational models deal with elementary mathematical constructs: functions of real-valued time and sequences of values, and it is ultimately the properties of these functions and sequences that we are interested in when we turn on the machine. Therefore, we must relate them to the properties the designer has guaranteed, and I believe I have done that.

8.1 Summary

This thesis reinforces the QDI circuit methodology by demonstrating that the commonly used atomic model (in which circuits are correct by construction) is provably implemented by the more accurate bulk-scale analog model, SPICE level 0. The model is *implemented* in the sense that any analog evolution agrees with some valid digital execution when observed using a fixed 4-threshold observation rule.

As our PRS is obtained from Martin Synthesis, we can assume the PRS is stable and noninterfering. The theory is otherwise general in the types of computations and PRS allowed, except for two assumptions we make (for now) to keep the basic theory as simple as possible: the observation theorem assumes mutually-exclusive disjuncts, and FenceCalc™ assumes a maximum logic-gate fanin of two. Other than these restrictions, we can rigorously verify any canonical PRS implementation, provided that, if necessary, enough delay elements are added so that the conditions required by the observation theorem can be met. If the conditions can be met, then FenceCalc™ will find sufficient conditions in time linear in the number of logic gates.

8.2 Improvement over SPICE Simulation: The Noise Budget

At best, a SPICE simulation checks a single analog behavior. Recall from Section 1.10.3 that a SPICE simulation has all of the following limitations:

1. Environment timing and all noise sources must be exactly known, or infinitely many simulations are required.

2. Device parameters must be exactly known, or infinitely many simulations are required.

3. SPICE has no way to account for its own numerical error.

By contrast, my method considers an infinite number of possible behaviors. We overcome limitation 1 using a noise bound η. Let $\eta_{FenceCalc}$ denote the η parameter supplied to FenceCalc^{TM}. We can trade some of this noide bound, $\eta_{Dynamical}$, for parameter uncertainty (as discussed in Section 7.5), overcoming limitation 2. Finally, we can trade additional noise bound, $\eta_{Numerical}$, for numerical error (as explained in Chapter 6) overcoming limitation 3. This leaves $\eta_{Dynamical}$ noise remaining for overcoming limitation 1.

In summary, $\eta_{FenceCalc}$ is the total **noise budget** that we are allowed, for a rigorous result. We compare the FenceCalc^{TM} calculation to the actual analog model, and bound differences in output current, yielding three categories of noise:

1. $\eta_{Dynamical}$ bounds total error due to model inaccuracy.

2. η_{Param} bounds total error from parameter uncertainty.

3. $\eta_{Numerical}$ bounds numerical error.

The total noise must fit into the noise budget $\eta_{FenceCalc}$:

$$\eta_{FenceCalc} = \eta_{Numerical} + \eta_{Dynamical} + \eta_{Param}$$

If we use the `eta-eff` command in FenceCalc^{TM} then we obtain the quantity $\eta = \eta_{FenceCalc} - \eta_{Numerical} = \eta_{Dynamical} + \eta_{Param}$. This is technically the quantity defined in Section 3.3 and used throughout most of this thesis.

8.3 Lessons Learned

When I started working on this problem I was originally hoping for a simple theory that would be easy for everyone to understand. As we discuss in Section 8.4, many of my results would clearly benefit from simplification, if possible. Nonetheless, in putting together complete, rigorous verifications of non-trivial circuits, I have identified certain relatively self-contained concepts which play important roles in reaching that result:

1. Until the underlying model is accurate enough, new failure modes are likely to arise. There is no substitute for **dealing directly with the underlying analog model**.

2. To make sense of behaviors in this underlying model, we need an **observation rule** (Section 3.5) that is immune to small noise and properly finds the events represented in an analog signal.

3. The observation rule must have four thresholds if the observation theorem is to be proved. This condition is necessary and sufficient.

4. Using the **Spatial Induction Principle (SIP)** (SIP, Section 4.4), we need only prove each gate correct independently of the other gates. When we do this, we can assume that the gate's inputs satisfy all imaginable safety properties.

5. When proving a gate's correctness, we can, using **extended stability**, (applied: Section 4.4.4, proved: Section 5) suppose that each input transition occured in isolation of other transitions on the same input.

6. It suffices to bound each transition by a pair of **fences** (leading and trailing) anchored at the point of transition initiation.

7. Numerically postulating a complete set of hypotheses that are proper fences is, asymptotically, a faster computation than a traditional non-exhaustive simulation. We need only a constant number of BFS passes over the circuit, regardless of the computation implemented by the circuit.

8. As illustrated in Section 7.3.2, sometimes a circuit is correct and verifiable, but we cannot verify it until we find the right observation thresholds. One general way to find the thresholds is to add delay elements until the circuit is verifiable, and then to use the continuous η metric discussed in Section 7.3.3 to fine-tune the thresholds, increasing the noise margin until we can remove the delay elements that we added.

8.4 Future Work

8.4.1 Simplifications

We have argued that an observation rule must have at least 4 thresholds if it is to lead to a provable observation theorem. This apparently necessary complexity pervades our entire theory, resulting in a total of 3 phases that must be considered in case analysis (in Section 4.4.6). Each of these cases has a corresponding calculation. While we have argued, intuitively, that each of these calculations is necessary, it would be nice if all thresholds and phases could be combined in some unified tabulation of some sort, with perhaps even a uniform intermediate operational model. My original approach was based on an intermediate operational model, but I abandoned that approach when it appeared to be too complicated.

8.4.2 Increased Noise Tolerance

Our results are based on the assumption that noise is small. While we can verify any circuit by adding delay elements, those elements must have small noise comparable to the noise allowed when those elements are used in a ring oscillator (Section 7.6). The question naturally arises as to whether we can handle more noise. Unfortunately, we cannot hope to come close to the mathematical limit, as that limit is likely to be a complicated fractal boundary[8]. However, it is also clear that our present results are not as close to that boundary as is possible. For example, techniques such as the one outlined in Section 7.7 could be implemented and evaluated.

8.4.3 Generalization

A calculator (such as a new version of FenceCalc$^{\text{TM}}$) should be written which extends hypothesis postulation to gates of higher fanins. More importantly, I have chosen a very simple type of underlying model and claimed that many deviations can be modeled as noise. However, the verifications I have produced do not succeed when the noise is too large. A theory should be developed to further characterize, through static analysis, the amount of noise that can be tolerated. When a new phenomenon (such as crosstalk) is encountered, it should be worked into the theory. Given such a phenomenon, there is perhaps an order in which solutions might be tried starting with the least-disruptive modification to the theory:

1. Attempt to include the phenomenon in the current-functions as accurately as possible.

2. Attempt verification with the phenomenon modeled as noise.

3. Add additional thresholds with corresponding new phases that constrain the signal more specifically than we have done.
4. Add additional analog state variables to the theory with corresponding notions of boundedness (and, possibly, new safety properties).

8.4.4 Calculation Speedups

Calculations using my method take time linear in the circuit size. Furthermore, as discussed in Section 8.2 we can do simulations that are not possible in a finite number of SPICE simulations. However, there are two problems which make my present method slow to use in practice: the large number of points required for numerical accuracy and large parameter uncertainty.

First, in my present implementation the calculation takes time linear in $1/\eta_{Numerical}$. Unfortunately the present noise budget is so small that thousands of points must be evaluated for each fence, even using the improved error bounding method (Section 7.4). The problem is that the underlying structure is a piecewise linear function, which requires many parameters. Perhaps other types of functions (e.g., splines) could be carefully designed to solve this problem, though I have found that even with splines of low degree, several segments are still necessary to verify a trivial circuit.

Second, and more importantly, my method can be very slow when applied to a large region of parameter space. As discussed in Section 7.5 we can, in finite time, verify a circuit over any closed region of parameter space by decomposition into many smaller patches. While this is much better than SPICE, it means that our calculation is now exponential in the circuit size (whenever the actual parameter noise exceeds the noise budget of a single calculation). Research is needed to find ways to reduce the amount of decomposition required. Perhaps symbolically-derived fences can be developed or the multi-scenerio fences proposed in Section 7.7 could be generalized.

Appendix A

Appendix: Failure-Causing Attributes of Analog Circuits

As discussed in Section 1.1, only a *rigorous* approach to analog verification allows us to rule out *all* possible failures, even those that have not been specifically discovered yet. Nonetheless, much work has focused on specific failure modes[39][41][23][38][7][56][51].

Some failures are caused by specific phenomena such as charge on internal nodes, crosstalk, and substrate phenomena that distort the physical behavioral model so much that new design techniques are required[56][51]. In some cases, a careful designer – armed with tools that simulate and analyze the specific phenomenon – can tolerate, correct, or even harness the phenomenon in its full strength. However, the only method which handles all phenomena consistently is to constrain the design in some way so that the phenomena can be minimized and treated as noise. The question for any particular phenomenon then becomes:

Can we treat this phenomenon as noise?

Recall that a rigorous approach refers to a specific model, and shows that no failures - i.e., no counterexample behaviors – are admitted in that model. We can view these failures as evidence of attributes of the physical model that distinguish it from a known good model (such as the atomic model). Therefore, we can answer the above question as follows:

A phenomenon is noise if its addition to the operational model
leads to no new failure examples.

Of course, this bottom-up reasoning is not an efficient way to arrive at a rigorous argument, so we do not employ it in our primary result, the observation theorem. However, for motivational purposes we attempt to consider various phenomena as individual

failure-causing attributes that we could (in principle) incrementally add to a model, incrementally revealing various types of failures.

The most significant such attribute is **non-atomicity**, which we now consider in its most abstract setting: as a property of concurrent composition.

A.1 Concurrent Composition Can Be Non-Atomic

Recall (Section 2.2.1) that the only requirements that a concurrent composition must satisfy are that

1. Each sequential projection is a sequence of firings on a single PR and
2. The effect of a (nonvacuously executed) PR $y{:=}v$ is the predicate on the Hoare triple $\{P\}y{:=}v\{Q\}$, which states that $Q = P$, except that the value of y is changed to v.

But these conditions do not imply atomicity. In general, each firing may consist of multiple events which are each individually interleaved into the execution. Thus, while it is true that y is changed to v, we cannot conclude (based on the abstract notion of concurrent composition) that this change happens in a single elementary step.

A.1.1 Spurious Ring Oscillators

Recall from Section 2.4.2 that spurious ring oscillation can occur in CMOS when a slewtime is too long. I now present SPICE simulation results illustrating that such failures really happen for the circuit shown in Figure 2.6. The amount of oscillation depends on the slewtime of the input voltage signal $x(t)$, as shown below:

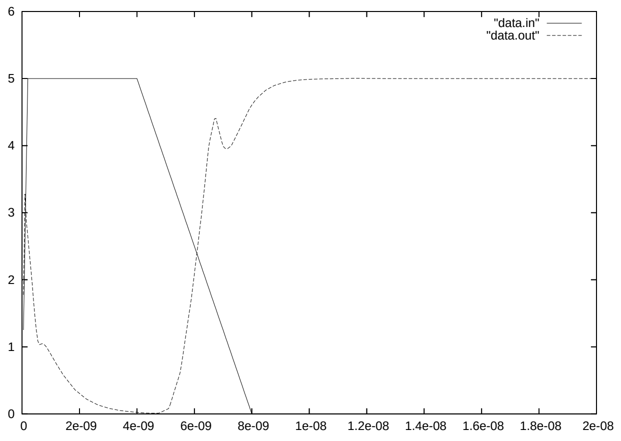


Figure A.1: Slightly Spurious Response to 4ns Input Rise Time.

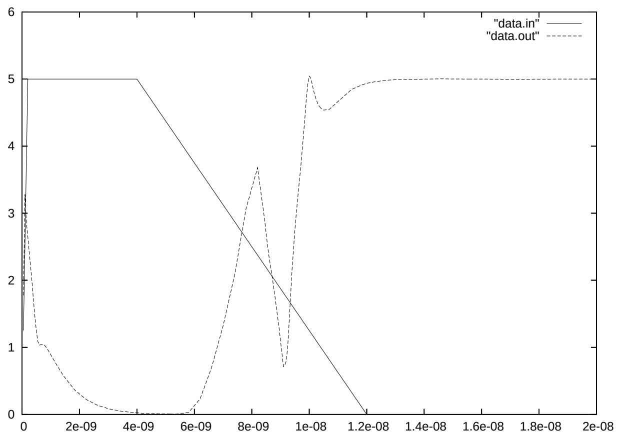


Figure A.2: More Spurious Response to 8ns Input Rise Time.

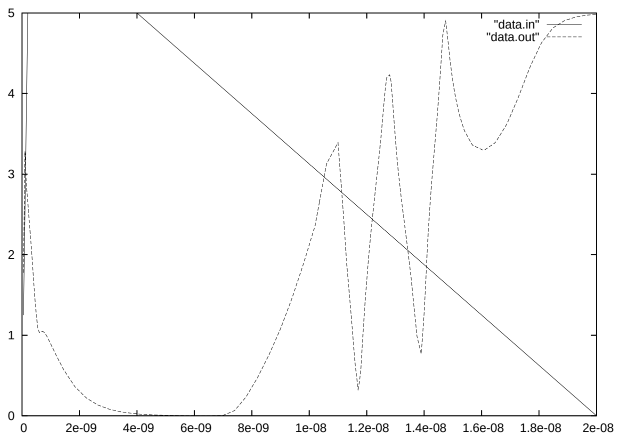


Figure A.3: Even More Spurious Response to 16ns Input Rise Time.

To reproduce the above waveforms, use the default 0.6μm model that comes with Berkeley SPICE 3 to simulate the following circuit definition in SPICE 3:

```
* transistor models
.include mos.cir
.OPTIONS ACCT NOPAGE

.global GND Vdd
Vgnd GND 0   0

* logic gate definitions

.SUBCKT inv1 in out
MP Vdd in out Vdd P12L5 L=1.2U W=30U
MN GND in out GND N10L5 L=1.0U W=10U
.ENDS

.SUBCKT weak in out
MP Vdd in out Vdd P12L5 L=1.2U W=3U
MN GND in out GND N10L5 L=1.0U W=1U
.ENDS

.SUBCKT celem2 a b out
M1 Vdd a p Vdd P12L5 L=1.2U W=30U
M2 p b out Vdd P12L5 L=1.2U W=30U
M3 GND a n GND N10L5 L=1.0U W=10U
M4 n b out GND N10L5 L=1.0U W=10U
.ENDS

* power supply
VDD Vdd GND PWL(0 0 0.1N 5)

* input waveform
Vin in GND PWL(0N 0 0.2N 5 4.0N 5 20.0N 0)

X1 in out3 out celem2
X2 out out2 inv1
X3 out2 out3 inv1

*staticizer
X4 out2 out weak

.TRAN 0.05N 20.0N
.PLOT TRAN V(in) V(out)
.END
```

A.1.2 Forms of Non-Atomic Concurrent Composition

The preceding examples of spurious oscillation illustrate that we can define physically accurate models[1] that are based on non-atomic concurrent composition. To see this, we note that a PR firing can be executed as a number of more elementary events; some possibilities are shown below:

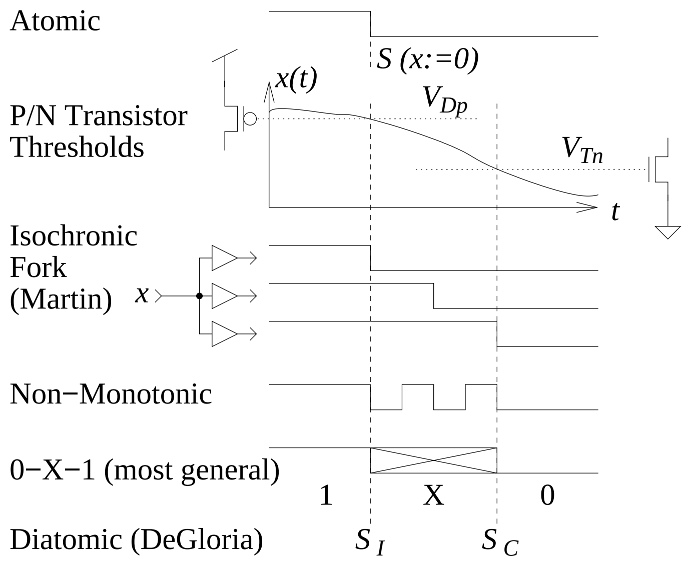


Figure A.4: Ways to describe the multiple events making up the execution of statement S.

The most general[7] event-based non-atomic model executes statement S as an **initiation event** S_I followed by the **completion event** S_C[38].

[1]Or, at least, models that are more physically accurate than the atomic model.

A.2 Nonoscillation

As mentioned in the introduction, all voltages could converge to a fixed point. Then no further oscillation if possible. This could occur either in the system as a whole or for some ring of gates.

This case is ruled out if the fixed point is non-attracting, which is guaranteed when the eigenvalues of its linearization are negative[8].

A.3 Leakage Drift

Dynamic nodes that are not actively driven might drift to a forbidden value. This problem is solved by using properly designed staticizers[1]. To rule out all counterexamples when the underlying model has leakage or current-noise, it is therefore clearly essential that the contributions of these staticizers also be included in the model.

A.4 Unexpected Slewtime

The shape of a transition into a gate affects the slewtime of the output of that gate. Therefore it is possible that, after a signal passes through a number of gates and its slewtime is incrementally improved, its shape becomes less and less desirable. A bad shape is one that can later be misinterpreted as multiple transitions or a transition of a longer slewtime than expected. It is easy to construct models in which this occurs.

Any analysis method which considers only the slewtime of transitions but not their entire shape is thus not rigorous: there exist circuit models for which it will fail. By contrast, my observation theorem and FenceCalc™ consider the entire transition shape. FenceCalc™ is capable of distinguishing the models which exhibit unexpected slewtime from those that do not.

Unfortunately, it appears unlikely that a verification based on a small number of parameters measuring transition shapes will succeed. I have tried hard to come up with such parameters, countable by the fingers on one's hand, that lead to a rigorous argument, but in the end only had success with piecewise-linear bounds containing dozens of points each.

A.5 Coupled Failures

The preceding examples can all be seen as failures on a single ring, i.e., they all result in some ring either oscillating too much or too little. However, in general, failures can occur in the coupling between rings without being revealed on any single ring.

For example, we could extend the spurious ring oscillator (Section 2.4.2) to have multiple points of synchronization. These extra synchronization points may control the overall oscillation so that the ring oscillates "just the right amount." However, the behavior still contains unsafe transitions, manifested as premature transitions on a given cycle of the behavior. To rule out such failures we need to relate the timing of the analog transitions to timing of the digital transitions. Our observation rule does this.

Appendix B

Appendix: Restoration Properties of CMOS

A CMOS circuit is a network consisting of MOSFET transistors. MOSFETS are three-terminal devices with insulated gates, and there are two varieties: NFETs (enabled by a positive gate voltage) and PFETS (enabled by a negative gate voltage). The drain-to-source currents i_n and i_p for NFETs and PFETs, respectively, are given by the formulas:

$$\begin{aligned} j_n = \frac{2}{k} i_n &= \max\left(v_{GSn} - v_{Tn}, 0\right)^2 - \max\left(v_{GDn} - v_{Tn}, 0\right)^2 \qquad \text{and} \\ j_p = \frac{2}{k} i_p &= \min\left(v_{GSp} - v_{Tp}, 0\right)^2 - \min\left(v_{GDp} - v_{Tp}, 0\right)^2, \end{aligned} \tag{B.1}$$

where v_{Tn}, v_{Tp}, and k are device parameters (n/p transistor thresholds and transconductance, respectively). We use j_n and j_p instead of i_n and i_p when the units of current are irrelevant.

B.1 Voltage Restoration

One can ask how the input voltage of a CMOS gate relates to its output voltage. Consider the simplest nontrivial CMOS gate, the inverter shown below:

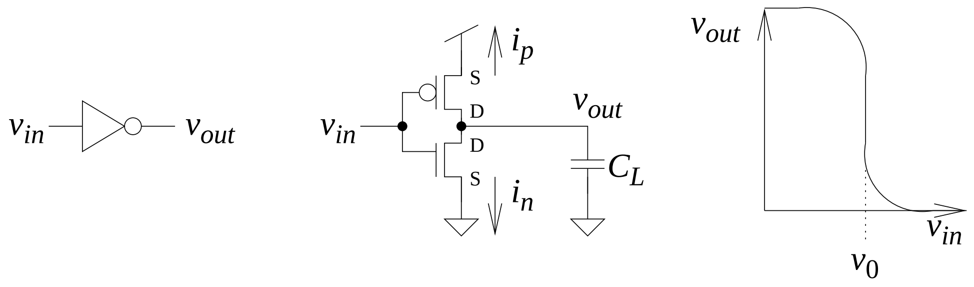


Figure B.1: Symbol, circuit, and Voltage Transfer Characteristic (VTC) for an inverter.

Defining $V_{Dp} = V_{DD} + V_{Tp}$, the MOSFET currents in an inverter circuit are:

$$\begin{aligned} j_n = \frac{2}{k} i_n &= \max\left(v_{GSn} - v_{Tn}, 0\right)^2 - \max\left(v_{GDn} - v_{Tn}, 0\right)^2 Tn \qquad \text{and} \\ j_p = \frac{2}{k} i_p &= \min\left(v_{GSp} - v_{Tp}, 0\right)^2 - \min\left(v_{GDp} - v_{Tp}, 0\right)^2 Dp. \end{aligned} \tag{B.2}$$

Consider for a moment the function

$$j_0(v_{in}) = \max(v_{in} - v_{Tn}), 0)^2 + \min(v_{in} - v_{Dp}), 0)^2. \tag{B.3}$$

If we assume that $V_{Dp} \geq V_{Tn}$, then there exists a unique v_0 such that

$$j_0(v_0) = 0. \tag{1}$$

To derive the voltage transfer graph in Figure B.1, we compute the steady-state value of v_{out} as a function of v_{in}, i.e., we assume that C_L can be ignored, and therefore $j_n + j_p = 0$. Combining this with (B.1) and (B.3), we obtain

$$j_0(v_{in}) = \max(v_{in} - v_{out} - v_{Tn}, 0)^2 + \min(v_{in} - v_{out} - v_{Tp}, 0)^2, \tag{2}$$

and solving for v_{out} in terms of v_{in} yields:

$$v_{out} = \beta(v_{in}) = \begin{cases} v_{in} - v_{Tn} \ldots v_{in} - v_{Tp} & , v_{in} = v_0 \\ v_{in} - v_{Tn} + \sqrt{j_0(v_{in})} & , v_{in} \leq v_0 \\ v_{in} - v_{Tp} - \sqrt{j_0(v_{in})} & , v_{in} \geq v_0 \ , \end{cases} \tag{3}$$

which is the curve drawn in Figure B.1.

Notice that β is locally constant near $v_{in} = 0$ and $v_{in} = v_{DD}$ and intersects the line $v_{in} + v_{out} = v_{DD}$ at these points. This means that β is *superattracting*: for any v_{in}, the sequence $(v_{DD} - \beta)^n(v_{in})$ quickly converges, i.e., when a DC voltage is fed through a small chain of inverters, it becomes 0 or v_{DD}. This well-known property is called *voltage restoration.*

Any other CMOS gate is equivalent to an inverter if all its inputs are tied together (because there is an Ohms Law for MOSFETs; see Appendix D.1), so the voltage transfer curve is the same as for an inverter. Beware, however, that the voltage transfer curve does not apply in the general case where inputs arrive at different times.

The voltage generation is best when NFETs are used as pulldowns and PFETs are used as pullups[1].

B.2 Slewtime Restoration

If steady-state arguments alone are used to verify a circuit, then voltage restoration is enough. But we have noted that in QDI, good input *transitions* must turn into good output *transitions.* The most basic property of these transitions which must be controlled is their slewtime.

One way to define *slewtime* is the time τ that it takes a transition signal to get from v_{Tn} (the 0 threshold) to v_{Dp} (the 1 threshold). One can ask the same questions of slewtime as one asked of steady-state voltage. Of course, the actual output slewtime is a function not just of the input slewtime, but also of the input signal shape. Thus we will have to eventually (to be rigorous) consider the relationship between *classes* of input signals and *classes* of output signals. For a crude intuitive analysis, however, let us (for now) assume a fixed input shape. In the limit of a very slow input we can ignore the gate output C_L so we can use the voltage transfer function β in order to compute the output signal as a function of the input signal. Suppose that when $\tau_{in} = 1$, the input shape is f_1. This leads to some output slewtime α:

$$\begin{aligned} v_{in,1}(t) &= f_1(t) \\ v_{out,1}(t) &= \beta \circ f_1(t) \\ \tau_{out}(\tau_{in}{=}1) &\overset{\text{def}}{=} \alpha. \end{aligned} \tag{B.4}$$

We can then scale the input shape f_1 in order to obtain different input slewtimes, and the corresponding output slewtimes:

$$\begin{aligned} v_{in,\tau_{in}(t)} &= f_1(t/\tau_{in}) \\ v_{out,\tau_{in}(t)} &= \beta \circ f_1(t/\tau_{in}) \\ \tau_{out}(\tau_{in}) &= \alpha \cdot \tau_{in}, \end{aligned} \tag{B.5}$$

i.e., for slow signals the slewtime transfer function approaches a linear asymptote of slope α. Note that $\alpha \ll 1$ due to the steepness of the output when it is between the 0 and 1 thresholds (in fact one could even set the thresholds and supply voltage such that $\alpha = 0$ due to the vertical segment in β. Beware of slow signals, however; see the next section, on "rough monotonicity").

For very quick input transitions, on the other hand, the input shape and speed is irrelevant due to the current integration at C_L. In this case the output is a piecewise exponential decay described by C_L discharging through the NFET with fixed gate voltage V_{DD}. The slewtime τ_{min} of this curve is the minimum possible slewtime.

The slewtime transfer function never achieves slope zero, so it is not actually superattracting, but it is close to superattracting, due to the small slope α and hence the proximity of the fixed point to the asymptote $\tau_{out} = \tau_{min}$, i.e., any slewtime in the vicinity of τ_{min} becomes τ_{min} after a small number of stages:

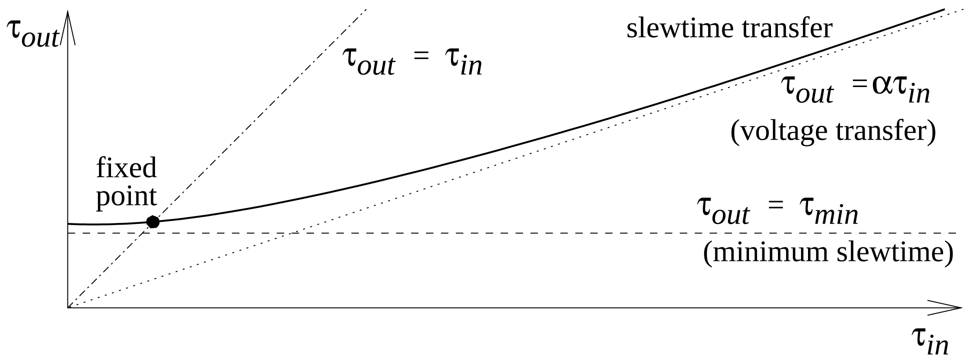


Figure B.2: Slewtime transfer function with asymptotes and fixed point.

Slewtime restoration intuitively says that given a long enough chain of delay elements, a fence pair of any slewtime is mapped to a fence pair of a fast slewtime. This can be checked for any given circuit technology (I have checked it in FenceCalc™for $V_{DD} = 5V_T$ with ¿1% noise). We use this version of slewtime restoration in Section 3.2.4 and Section 7.6.

B.3 Rough Monotonicity

Clearly the preceding characterization of a slewtime transfer function depended on the actual shape of the input transition signal (even though we ignored this fact for the purpose of developing an intuition). In addition to the slewtime of a signal, we may also be interested in some property of the signal's shape, such as monotonicity.

Of course, no physical signal is actually monotonic, due to the presence of noise. Instead, we characterize signals as "roughly monotonic." In particular, if we know that an input signal is not obnoxiously squirming about between the 0 and 1 thresholds for a prolonged period of time, but rather is making its way towards its target at a steady pace, then we might get much better slewrate restoration (not to mention glitch avoidance, if the gate happens to be combinational).

A definition of "rough monotonicity" should constrain the shape of a transition from the moment it *first* crosses the 0-threshold. Clearly it should not affect the output before this time, but its affect on the output must be considered from this time on (due to the memory of C_L). If the first constraint on the input signal has the form of an *upper bound shape*, then we can guarantee (by upper-bounding the current into C_L) that the output cannot cross the threshold taking it away from its previous value for a certain *output dislodge time*.

Once the output dislodge time has elapsed, the “output slewtime” stopwatch has been started, so from this point on some sort of lower bound is required of the input signal, enabling the output slewtime to be upper-bounded (by lower-bounding the current into C_L):

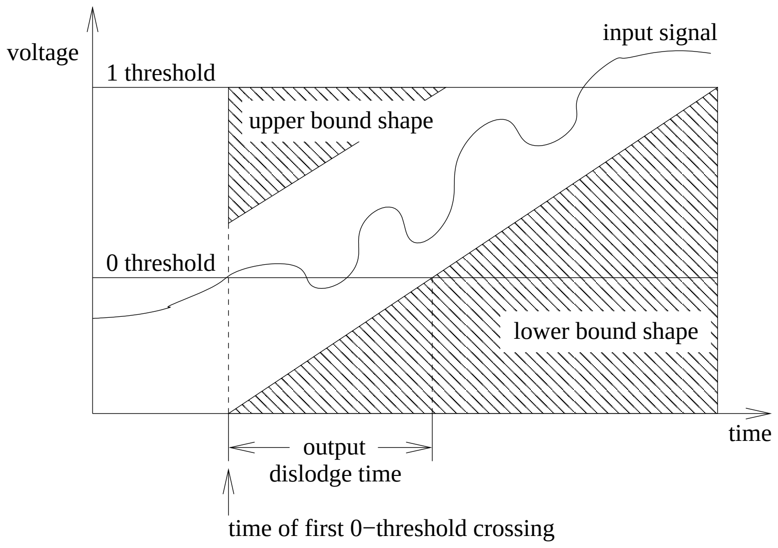


Figure B.3: A roughly monotonic input transition signal.

Notice that the above arguments are actually easier to apply to dynamic gates than to combinational gates because in dynamic gates the opposing guard has been cut off before the input transition begins. v

Appendix C

Appendix: Fences

C.1 Motivation

As discussed in Section 1.6 we are confronted with the problem of how to bound waveforms $x(t)$. Fences[8] allow us to bound the $x(t)$ – solutions to DEs (differential equations) – without having to solve the DEs exactly.

In contrast to DE solutions, a fence can have virtually any mathematical form so long as it is differentiable. Given a differential equation $x' = f(t, x)$, a fence need only satisfy the following condition (depending on whether it is an upper or lower fence):

fence type	**condition**
lower	$l'(t) < f(t, l(t))$
upper	$u'(t) > f(t, u(t))$

Figure C.1: A Fence Need Only Satisfy a Local Condition

The **fence theorem**[8] (Section 1.6) states that if a solution is above a lower fence or below an upper fence, then it remains that way for all time. The remainder of this appendix is devoted to examples of fences for various differential equations.

C.2 Trivial Example

Consider the differential equation $x' = -x$. The function $u(t) = 1$ is an upper fence:

$$u'(t) = 0 > -1 = -u(t) = f(t, u(t)) \tag{C.1}$$

Therefore once a solution is less than 1, then it is forever less than 1.

C.3 Example: DE with no solution formula

The differential equation $x' = x^2 - t$ has no solution formula[8]. Nonetheless, the functions $x^2 - t = c$ are fences (whose type depends only on c and the branch of square root taken)[8].

Notice that this fence has a completely different form from the DE solution itself: the fence has an elementary formula, while the DE solution can't even be expressed in terms of elementary and algebraic functions. This is of course the main point about fences: we avoid having to find the exact mathematical form of the DE solutions. This will be true of all the following examples in this Appendix.

C.4 Graphical Circuit-Waveform Example

Consider the simple example of an inverter in the lumped capacitance model presented in Section 3.2.1. Suppose for simplicity that the input $x(t)$ is a picewise-linear ramp function as shown below:

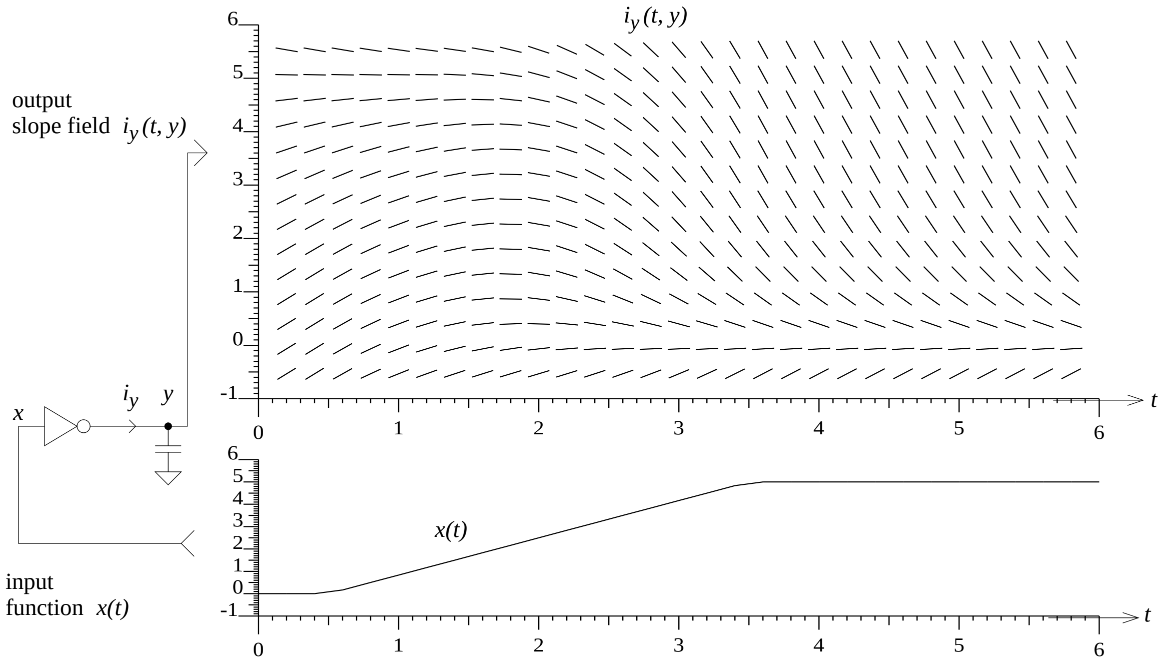


Figure C.2: Slope Field for Inverter Output Waveform.

We wish to bound the output $y(t)$ without actually having to solve the DE. We can easily visualize possible output waveforms by tracing the slope marks in Figure C.2 for various initial conditions. We get some sort of falling transition, as expected.

In Section B.2 we suggested that the slewtime of a falling transition has a lower limit given by the Voltage Transfer Characteristic (VTC) and an upper limit given by an RC delay. To give a rigorous example we show that the former type of function is a **leading fence** (i.e., lower fence for a falling transition) and the latter type of function is a **trailing fence** (i.e., upper fence for a falling transition), as shown below:

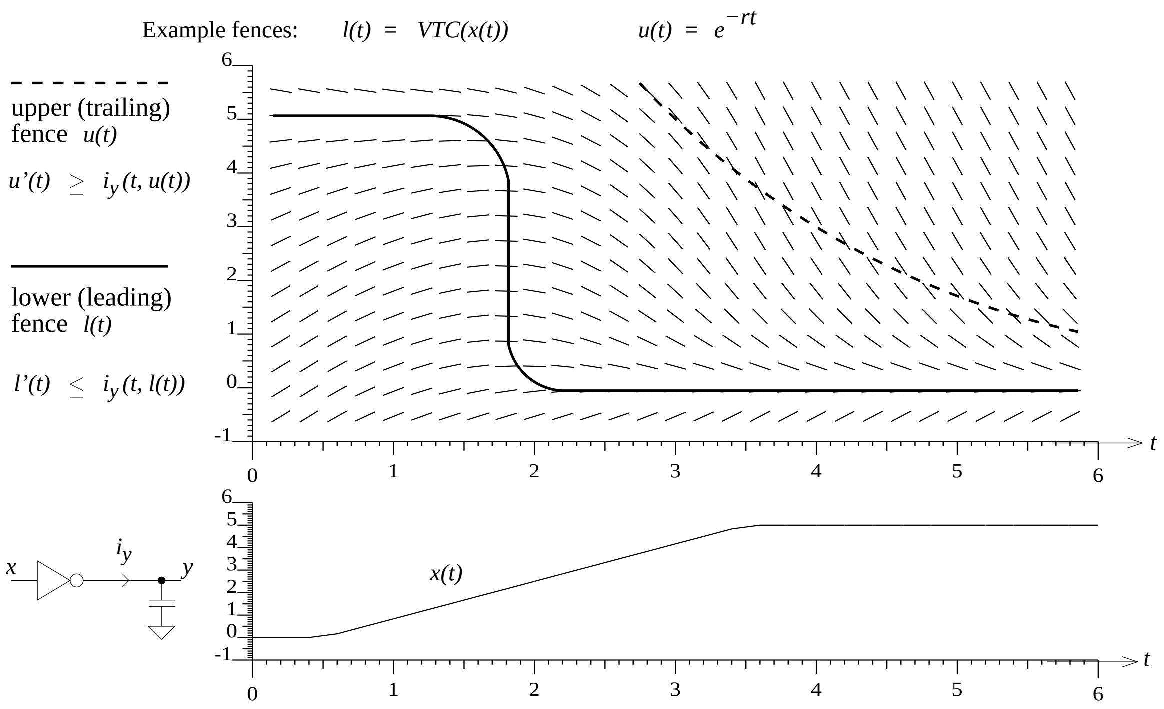


Figure C.3: Upper and Lower Fences for Inverter Output Waveform.

C.4.1 Leading Fence Example: VTC

By definition the Voltage Transfer Characteristic (VTC) is the steady-state output voltage, i.e., it satisfies $i_y(t, VTC(x(t)) = 0$.

Let $l(t) = VTC(x(t))$. The fence condition follows from the chain rule and the monotonicity of $VTC(x)$ and of $x(t)$:

$$l'(t) = \frac{\mathrm{d}VTC(x(t))}{\mathrm{d}t} = VTC'(x) \cdot x'(t) \leq 0 = i_y(t, VTC(x(t)) = i_y(t, l(t)) \qquad \text{(C.2)}$$

Thus for any circuit with negative-monotonic VTC, then the VTC is a lower fence on the response to any positive-monotonic input.

C.4.2 Trailing Fence Example: Exponential Decay

We now show that for some constant r and any constant C, the functions $u(t) = Ce^{-rt}$ are upper fences. We make the following two assumptions:

1. The rising input has reached (and remains above) V_{Dp}. Thus our fence has a left endpoint.

2. C is such that the fence is initially less than $2V_{DD}$.

Our first assumption gives us

$$-i_y(t, u(t)) \geq -i_y(x = V_{Dp},\ y = u(t)). \tag{C.3}$$

Combining this with our second assumption and the concavity of $i_y(y)$, we have:

$$-i_y(t, u(t)) \geq -i_y(x = V_{Dp},\ y = 2V_{DD})\frac{u(t)}{2V_{DD}}. \tag{C.4}$$

Thus we have a positive r for which the fence condition

$$i_y(t, u(t)) \leq -ru(t) = -u'(t) \tag{C.5}$$

is satisfied.

C.5 Piecewise Linear Fences (FenceCalc™)

FenceCalc™ uses piecewise-linear functions as fences. Clearly these functions are not *themselves* solutions to any continuous DE. However they are accurate fences because they satisfy the fence conditions.

C.6 Higher-Dimensional Fences: The Generalized SIP

In my observation theorem fences allow us to bound a *single* waveform *assuming all other waveforms are bounded.* However, the existing mathematical theory of fences does not naturally give a way to bound *all* waveforms *in the absence of given bounds.*

I designed the Spatial Induction Principle (SIP) (Section 4.4) to solve this problem. According to the SIP, if each signal is individually bounded (assuming everything around it is weakly bounded), then all signals are bounded.

The SIP given in Section 4.4 is intertwined with my observation theorem because my bounds are fences over some time intervals and derived from the observation rule over other time intervals. However, as an exercise we can extract an abstract mathematical result from the SIP, as follows:

Theorem 7 (Generalized SIP) *Consider any multi-dimensional continuous ordinary DE f, with differentiable bounds $l_y(t)$ and $u_y(t)$ for each variable y. Consider any solution s and suppose that the following two conditions hold:*

i. Suppose that each variable y satisfies the following **spatial inductive step***. Let $x_1, \ldots, x_k$ denote the variables other than y. Suppose each is associated to any integrable function (not necessarily DE solutions) bounded* **weakly** *by the corresponding l and u bounds. Consider any spatially local solution p_y such that $p'_y(t, x_1, \ldots, x_k, y) = f(t, x_1, \ldots, x_k, y, p(t, x_1, \ldots, x_k, y))$. The spatial inductive step is that p_y then satisfies the bounds strongly.*

ii. Suppose the following initial condition holds: $l_y(t_0) < s_y(t_0) < u_y(t_0)$ for all y.

Then $l_y(t) < s_y(t) < u_y(t)$ for all $t \geq t_0$.

which is proved similarly to the SIP proved in Section 4.4.

Theorem 7 may well be the first published generalization of fences to higher dimension.

An interesting mathematical question is whether (under additional Lipschitz-like conditions, perhaps) the word "weakly" in the above theorem can be changed to the word "strongly". Clearly this would be a stronger mathematical result, but of course there is no difference for the practical application considered in this thesis.

Appendix D

Appendix: Current-Bounding Calculations

A CMOS inverter has two transistors connected in parallel to the output; therefore the output current i_o is simply the sum of the NFET and PFET currents, namely:

$$\begin{aligned} i_o &= i_p - i_n \qquad \text{(D.1)} \\ &= k_p\Big[\mathrm{SM0}(V_{DD} - v_i - v_{Tp}) - \mathrm{SM0}(vo - v_i - v_{Tp})\Big] \\ &- k_n\Big[\mathrm{SM0}(v_i - 0 - v_{Tn}) - \mathrm{SM0}(v_i - v_o - v_{Tn})\Big] \end{aligned}$$

(using the SM0 notation of Section 2.5.3). In general, when a number of elements are connected in parallel between supply rails and the output, the output current is the sum of the individual elements' output currents.

In a general CMOS gate, however, there are series transistors. In Section D.1 we show how to exactly find the current-function for an arbitrary network of NFETs in the case where all inputs are connected together. Of course, all-PFET networks are similar. Then in Section D.2, we generalize our equations to handle different inputs for the 2-input case. In Section D.3 we further generalize to the case where the inputs are not known exactly, but are bounded.

Finally, in Sections D.4-D.5 we discuss DC bounding and error bounding used in FenceCalc$^{\text{TM}}$.

D.1 Generalized Ohm's Law

In this section we show how *any* network of NFETs with a common input signal is equivalent to a single NFET. We show this by demonstrating that under a **Change Of Variables**

(COV), an NFET satisfies Ohm's law. Therefore there is an equivalent circuit for any network.

The idea of changing variables and finding an equivalent resistor netork has been used in the analysis of linear circuits circuits[34]. For example, in the circuit below we begin by assuming that each signal x has the form xe^{st}. By making a change of variables from xe^{st} to x, we can view any circuit element as a resistor as shown below:

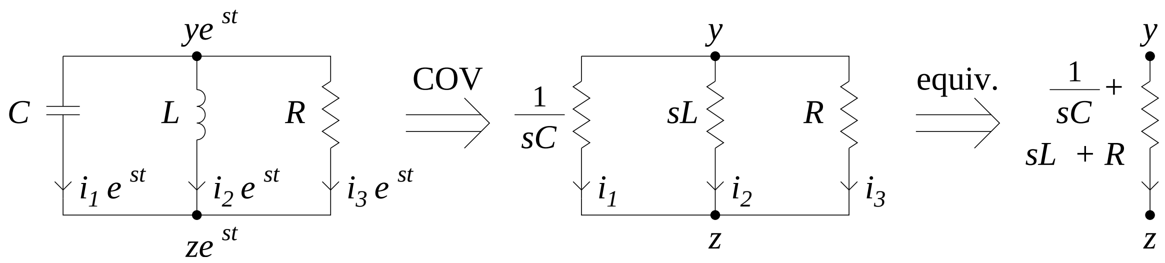


Figure D.1: Analysis of a linear circuit using generalized Ohm's law.

The change of variables is allowed because the new variables satisfy KCL, KVL, and Ohm's law. In this example, KVL is implicit, KCL holds by linearity of the change of variables, and Ohm's law holds for each component. For example,

$$i_1 = sC(y - z) \tag{D.2}$$

holds because it follows directly from the original component equation, namely,

$$i_1 e^{st} = C\frac{\mathrm{d}}{\mathrm{d}t}\left[(y - z)e^{st}\right]. \tag{D.3}$$

I now show that we can sometimes apply the same technique to *nonlinear* circuits by using a *nonlinear* change of variables. Consider any network of NFETs with identical V_T and all inputs connected to the same node a. This situation would arise in any multi-input gate if all input transitions were identical and arrived at the same time. We make a change of variables to our NFET network for the voltages only: change all x to $-\mathrm{SM0}(a - x - V_T)$, as shown below:

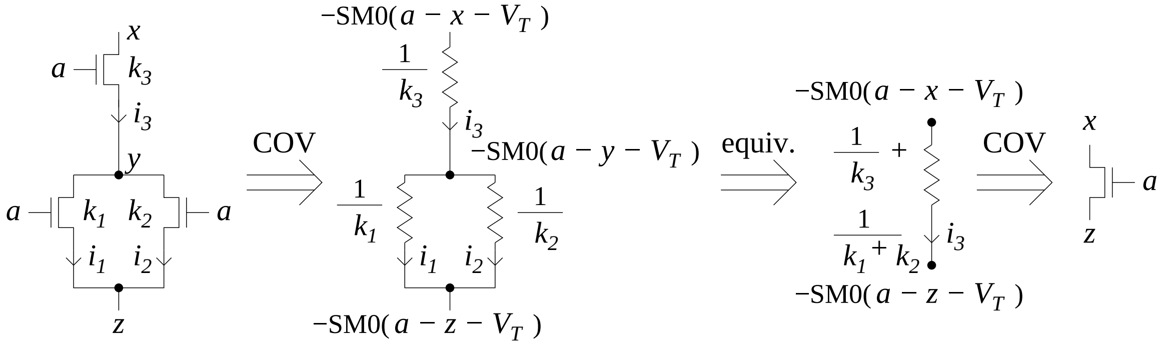


Figure D.2: Analysis of a common-gate network of NFETs using generalized Ohm's law.

As in the linear circuit example, this change of variables is legal because KVL, KCL, and Ohm's law are satisfied. KVL is again implicit, and KCL has not changed because we only changed the voltages. Ohm's law holds, as before, because it gives us identical equations in both circuits. For example, the equation for circuit element number 1 (in both circuits) is as follows:

$$i_1 = k_1\Big[\mathrm{SM0}(a - z - V_T) - \mathrm{SM0}(a - y - V_T)\Big] \tag{D.4}$$

Finally, we reduce the resulting resistor network to an equivalent circuit consisting of a single resistor and convert back to a circuit consisting of a single NFET in the original variables.

D.2 Current-Function for Two-Input Gates

In the preceding section we showed how to find the current-function for an arbitrary NFET network *if all inputs are the same.* In general, of course, that assumption does not hold. Let us now derive a more general current-function for a chain of 2 transistors in series. Let a and b represent the input voltages less the transistor thresholds. Without loss of generality, we can assume we have two NFETs connecting an output z to ground as shown below:

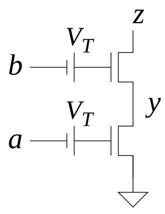


Figure D.3: Analysis of two NFETs in series.

Now make the assumption that both transistors have the same transconductance k. This is a reasonable assumption because series transistors are typically cut out of a single strip of diffusion and hence all have the same width. By KCL the two transistors have the same current as follows:

$$\begin{aligned} i/k &= \mathrm{SM0}(a) - \mathrm{SM0}(a{-}y) \\ &= \mathrm{SM0}(b{-}y) - \mathrm{SM0}(b{-}z). \end{aligned} \tag{D.5}$$

Next we solve for the unknown voltage at y by collecting the terms containing y into the following expression:

$$\begin{aligned} j &\overset{\mathrm{def}}{=} \mathrm{SM0}(a{-}y) + \mathrm{SM0}(b{-}y) \\ &= \mathrm{SM0}(a) + \mathrm{SM0}(b{-}z). \end{aligned} \tag{D.6}$$

Notice that we can compute j immediately; it is simply $\mathrm{SM0}(a) + \mathrm{SM0}(b{-}z)$.

Now we continue to solve for y, but we don't know how to evaluate SM0(a−y) or SM0(b−y) yet because we don't know if $a \geq y$ or $b \geq y$. Therefore we do a case analysis (i.e., we look for solutions in each quadrant), as follows:

$$\begin{cases} (a-y)^2 = j & \text{, if } (a \geq y) \wedge (b < y) \\ (b-y)^2 = j & \text{, if } (a < y) \wedge (b \geq y) \\ (a-y)^2 + (b-y)^2 = j & \text{, if } (a \geq y) \wedge (b \geq y) \end{cases} \tag{D.7}$$

Notice that we need not consider the case where $(a < y) \wedge (b < y)$, as in this case the current is zero, which we can see by choosing $a = b = y$, a case already covered.

Solving for y, we have:

$$\begin{aligned} 2y &= \begin{cases} 2a - 2\sqrt{j} & \text{, if } (a \geq a - \sqrt{j}) \wedge (b < a - \sqrt{j}) \\ 2b - 2\sqrt{j} & \text{, if } (a < b - \sqrt{j}) \wedge (b \geq b - \sqrt{j}) \\ a + b - \sqrt{2j - (a-b)^2} & \text{, otherwise} \end{cases} \\ &= \begin{cases} 2a - 2\sqrt{j} & \text{, if } \big((a-b)^2 > j\big) \wedge (a > b) \\ 2b - 2\sqrt{j} & \text{, if } \big((a-b)^2 > j\big) \wedge (a < b) \\ a + b - \sqrt{2j - (a-b)^2} & \text{, if } \big((a-b)^2 \leq j\big) \end{cases} \end{aligned} \tag{D.8}$$

Ultimately we want the current-function, a function of this y. Referring back to Equation D.5, we obtain the following:

$$\frac{2i}{k} = \begin{cases} -2\mathrm{SM0}(b-z) & \text{, if } \big((a-b)^2 > j\big) \wedge (a > b) \\ -2\mathrm{SM0}(a) & \text{, if } \big((a-b)^2 > j\big) \wedge (a < b) \\ \mathrm{SM0}(a) - \mathrm{SM0}(b-z) + \mathrm{SM0}(b-y)^2 - \mathrm{SM0}(a-y) & \text{, if } \big((a-b)^2 \leq j\big). \end{cases} \tag{D.9}$$

Letting $j_\Delta \overset{\text{def}}{=} \mathrm{SM0}(a) - \mathrm{SM0}(b-z)$, we can simplify the last case, as follows:

$$\begin{aligned} \frac{2i}{k} &= \mathrm{SM0}(a) - \mathrm{SM0}(b-z) + \mathrm{SM0}(b-y)^2 - \mathrm{SM0}(a-y) \\ &= j_\Delta + (b-y)^2 - (a-y)^2 \\ &= j_\Delta + (b^2 - a^2) - 2(a-b)y \\ &= j_\Delta + (a-b)\big(2y - (a+b)\big) \\ &= j_\Delta + (a-b)\sqrt{2j - (a-b)^2}. \end{aligned} \tag{D.10}$$

In summary, the current-function for a two-transistor series chain is as follows:

$$\frac{2i}{k} = j_\Delta + \begin{cases} -j & \text{, if } \Big((a-b)^2 > j\Big) \wedge (a > b) \\ j & \text{, if } \Big((a-b)^2 > j\Big) \wedge (a < b) \\ (a-b)\sqrt{2j-(a-b)^2} & \text{, if } \Big((a-b)^2 \le j\Big). \end{cases} \tag{D.11}$$

D.3 Input-Cube to Output Bound

The calculations described in Section 6.5 require us to compute the maximum or minimum possible value of a gate's current-function given that the inputs are contained in some cube C, and the output has some value v_o. Without loss of generality, consider the problem of finding the minimum (or most negative) current. When $v_0 \in [0, V_{DD}]$, the current-function is anti-monotonic in the inputs, so the answer is trivial: we always choose the corner of C which maximizes i.

Unfortunately, the starting points of our leading fences and the ending points of our trailing fences fall outside $[0, V_{DD}]$, where the current-function does not have constant monotonicity. In that case we have to solve an optimization problem.

For our current-function the extremum can probably be found exactly. However, it suffices to compute a conservative lower bound on the current function, so that is what FenceCalc$^{\text{TM}}$ does. To bound Equation D.11, we evaluate its derivatives[1] with respect to a and b at the corners of C. Since Equation D.11 has fixed concavity with respect to the inputs, we can bound the current by evaluating (at each corner) the linearization of i at each other corner, and taking the minimum.

The aforementioned bound is overconservative, but can be improved by subdividing the cube. About 8 binary subdivisions are required to successfully verify the circuits described in this thesis. At present this is a slow computation, but it can be improved by selective subdivision and/or further symbolic analysis.

D.4 Calculation of DC bounds

To calculate the DC bounds we find constant-valued fences. It suffices to find voltages such that for each gate, when the inputs are within the bounds, the output current at the upper bound is negative and the output current at the lower bound is positve. The DC version of the `update` method (Section 6.4.2) postulates these bounds quickly using

[1]The exact formula is easy to derive.

Newton's method[57].

D.5 Euler Error

Any implementation of Euler's Error has two sources of error: rounding error and the error of the method itself. We focus on the later, which is much more signficant if the computer has reasonable floating-point accuracy[8].

Let h denote the `stepsize` used. Assume a fixed differential equation. Then, as a function of h, there exists an **error bound** ϵ_h such that the piecewise linear Euler solution u_h satisfies the **slope error equation**

$$\left|u'_h(t) - f\Big(t, u_h(t)\Big)\right| \leq \epsilon_h. \tag{D.12}$$

One generic, conservative choice[8] of ϵ_h is as follows:

$$\epsilon_h = h(P + KM) \tag{D.13}$$

which tends to zero linearly in h. The constants M, P, and K are defined as follows:

$$\begin{array}{rcll} \sup|f| & \leq & M & \text{volts / second} \\ \sup|\dfrac{\partial f}{\partial t}| & \leq & P & \text{volts / second}^2 \\ \sup|\dfrac{\partial f}{\partial x}| & \leq & K & \text{second}^{-1} \end{array} \tag{D.14}$$

assuming time is measured in seconds and trajectories in volts.

The `eta-eff` command in FenceCalc™ provides two rigorous methods of computing the Euler error: a static method and a dynamic method. The user can choose whichever gives a smaller error estimate. The **static method** uses the above formula, and the M, P, and K above are derived from the circuit equations and DC bounds. That derivation is extremely boring and takes 10 pages (which we omit) to describe in full detail. The importance of this result is not so much that we can compute M, P, and K, but that they exist in theory, so that the error decreases with h. From a practical point of view, there are better methods.

The **dynamic method** appears to be more practical, as it does not require h to be as small. This method directly analyses the Euler-generated function over each time interval $[hi, hi{+}h]$. Since the inputs are piecewise linear, we can find the input-cube that applies *over* *the entire interval* $[hi, hi+h]$ by simply taking the union of the input cubes at the endpoints of the interval. We then use our existing bounding function over this cube and compare it to what the ordinary Euler's method did. Notice that these extra calculations need only be done at the end of the verification after all the postulation is complete. Therefore the advantages of the dynamic method (larger h) far outweigh its disadvantages (twice the work in the very last phase of verification).

Appendix E

Appendix: The KAVL Interactive Command Language

As discussed in Section 1.8, FenceCalc™ is driven by the interactive command language `KAVL`. In Section E.1 we begin by discussing the standard `KAVL` script that I have developed in order to attempt to verify any circuit. Then, in Section E.2, I give the general form of all `KAVL` commands, including those used in the script.

E.1 Universal Verification Script

I have written a **universal verification script** in KAVL. This script applies the sequence of verification steps presented in Section 6.2. The script can be applied to any canonically-generated PRS circuit, and if it terminates successfully, the conditions of the theorem hold, and therefore the circuit is rigorously verified:

```
# load the circuit
load-kast syncloop.kast .0001


#########################
# postulate hypotheses

#infer completion thresholds
epsilon calc

#calculate all hypotheses
bfs-tol 1.0e-4
bfs-limit 10
bfs dc
bfs tran init


#########################
# check the result

#make sure epsilons are consistent with each other
epsilon check

#verify all hypotheses
bfs-tol 1.0e-6
bfs check

#check transmission-safeness
slew-check

#evaluate final noise tolerance of the proof,
#by subtracting all calculation error sources from "eta" for each node.
round-err 1.0e-2  #includes effect of bfs-tol
eta-eff verbose

#exit if successful
quit
```

If the script fails, then the circuit can always be modified to make the script succeed. First, the initiation thresholds are modified (with the corresponding changes to completion thresholds automatically inferred by the script). If this fails, then the circuit must be changed: first sizing is changed, then delay stages are added as a last resort.

E.2 List of Existing Commands

Command descriptions in `typewriter type` are directly from the `help` command in FenceCalc™.

```
help -- display list of commands
save <filename> -- save previously-executed commands
source <filename> -- execute commands from a file
quit -- exit the command loop
gnuplot ... -- manipulate active plots
digits [<val>] -- display/set floating-point output precision
bfs-tol [<val>] -- display/set exit tolerance ('delta') for 'bfs' command
bfs-limit [<val>] -- display/set limit to number of 'bfs' passes
vtc-lo [<val>] -- display/set VTC lowest input voltage
vtc-hi [<val>] -- display/set VTC highest input voltage
round-err [<val>] -- display/set max absolute rounding error R
auto-display [<val>] -- display/set display plots on creation (see 'gnuplot')
load-kast <file.kast> <stepsize-seconds> -- load KAST from <file.kast>
setup -- display general hypothesis-system info
nodes -- list circuit nodes
fanin <node> -- describe gate driving <node>
fanout <node> -- describe gates driven by <node>
dc ... -- manipulate dc hypothesis
tran ... -- manipulate transition hypothesis
eta ... -- manipulate noise-bound on nodes
epsilon ... -- manipulate initiation/completion thresholds
epsiset ... -- set multiple 'epsilon I' thresholds
bfs [<modes>] -- calculate all system hypotheses using BFS
slew-check -- verify all slewtime constraints
handshakes [verbose] -- compute minimum delay for all handshakes
vtc <node> [<input-vnoise>] -- plot VTC of 1-input gate driving <node>
input-cube <tran> -- plot input intervals for <tran>
eta-eff [...] -- calculate effective eta (= eta - Euler err)
# [...]       -- comment (ignored by the program)

help <command> -- extended help is available for the following commands:
eta-eff vtc bfs epsiset epsilon eta tran dc gnuplot
```

E.2.1 The `epsilon` Command

```
epsilon ... -- manipulate initiation/completion thresholds

epsilon i             -- display initiation thresholds for all transitions
epsilon c             -- display completion thresholds for all transitions
epsilon <tran> [i|c]  -- display initiation/completion threshold(s) for <tran>

epsilon <tran> i|c <value> -- set "i" or "c" threshold of <tran> to <value>
epsilon        i|c <value> -- set all "i" or "c" thresholds to <value>

epsilon <tran> check       -- check dc consistency of "c" thresh. of <tran>
epsilon        check       -- check dc consistency of all "c" thresholds

epsilon [<tran>] calc [<margin>]
   automatically selects consistent "c" thresholds for <tran>
   (or all transitions). A value a factor <margin> larger than
   the minimum is chosen. If no <margin> is given, 1.1 is used.
```

E.2.2 The epsiset Command

```
epsiset ... -- set multiple 'epsilon I' thresholds


epsiset <tran1> [<tran2> ...] <value12>
       [<tran3> [<tran4> ...] <value34>
       [...]

is equivalent to

epsilon <tran1> i <value12>
epsilon <tran2> i <value12> [...]
epsilon <tran3> i <value34>
epsilon <tran4> i <value34> [...]
:
```

E.2.3 The eta Command

```
eta                 -- display "eta" of all nodes
eta <value>         -- set "eta" of all nodes to <value>
eta <node>          -- display "eta" of <node>
eta <node> <value> -- set "eta" of <node> to <value>
```

E.2.4 The eta-eff Command

```
eta-eff [...] -- calculate effective eta (= eta - Euler err)

eta-eff [<node>] [<subdiv>] [verbose]

If <subdiv> is 0 or not given, use general Euler bound,
h(P + K*M). This formula depends only on the DC bounds and
input hypotheses, but is overconservative.

If <subdiv> is an integer >= 1, then evaluate error
directly on partial output hypotheses. Use larger <subdiv>
for more accuracy.

The arguments to "eta-eff" can be given in any order.
```

E.2.5 The `vtc` Command

This command is intended as a test of current-bounding and is not used in verification.

```
 vtc <node> -- plot VTC of 1-input
gate driving <node>.

    Output (current) noise is taken into consideration.
    The VTC consists of two curves, "upper" and "lower",
    corresponding to the range of possible steady-state
    voltage for a noise of "eta(<node>)".


vtc <node> <input-vnoise> -- An extended form which takes both input
                             and output noise into consideration.

    For each "<vin>" value, assumes only that the input voltage is
    contained in the interval "<vin> +/- <input-vnoise>".
```

E.2.6 The `dc` Command

```
dc <node>               -- display current dc hypothesis for <node>
dc <node> <lo> [<hi>]   -- set hypothesis for <node>
dc <node> calc [acc]    -- calculate hypothesis for <node> from fanin hypotheses
                           To weaken the existing hypothesis, use "acc".
```

E.2.7 The `tran` Command

```
The "tran" command allows manipulation of transition hypotheses.
A transition <tran> is specified as <node><+|->.
```

E.2.7.1 Displaying information about transition hypotheses

```
tran <tran>                  -- display current hypothesis for <tran>

tran <tran> check            -- check and display containment at <tran>,
                                showing both current and calculated hypotheses,
                                but do not change the current hypothesis.
                                (see the "bfs check" command)

tran <tran> partial [raw]    -- show partial-hypothesis fences for <tran>
                                (same as hypothesis for 1-input gate)
                                and (if indicated) "raw" data before
                                threshold-crossing.
```

E.2.7.2 Changing the transition hypotheses

```
tran <tran> <gap> [<slope>] -- set <tran> hyp. to 3-piece fences
                               with delay of "gap" between leading & trailing.

tran <tran> calc [acc]      -- update hypothesis for <tran> based on fanin
                               hypotheses. To weaken the existing
                               hypothesis, use "acc"
                               (for all transitions, see the "bfs" command).
```

E.2.8 The `bfs` Command

```
bfs [<modes>] -- calculate all system hypotheses using BFS
```

E.2.8.1 General Form

```
bfs [init|acc|check] [dc|tran] [verbose-level]

When used without the "check" option, the bfs command
causes a BFS on the circuit, which continues until either

    <containment-distance>  <  <bfs-tol>     (see bfs-tol command)

or <bfs-limit> passes are reached (see bfs-limit command).

A BFS actually consists of one standard BFS starting from the
graph's basis (for input nodes) followed by a "synchronizing BFS"
starting from the graph's cutset.
```

E.2.8.2 The `bfs check ...` Form

```
This form of the command is used to check the results of a BFS.
It updates the hypothesis of every transition in the circuit,
returns the worst containment, and indicates whether this
is within tolerance.

As a check on the BFS algorithm itself, this command simply
iterates over all transitions in the circuit rather than doing a BFS.
After a "bfs check ..." command the existing hypotheses are unchanged.
```

E.2.8.3 Options for the bfs Command

```
[tran|dc]  = dc      -- update dc hypotheses
             tran    -- update transition hypotheses
             <empty> -- update dc hyp., then update transition hyp.

[init|acc|check] = init    -- do bfs without hypothesis accumulation
                              (i.e., replace existing hypotheses)
                   acc     -- do bfs with accumulation
                   check   -- do "bfs check" instead of bfs (see above)
                   <empty> -- do one "init" and one "acc".

[verbose-level] =  0  -- no output
                   10 -- print phase information (ac only)
                   15 -- print phase information
        <empty> or 20 -- print step  information (ac only)
                   25 -- print step  information
```

E.2.9 The gnuplot Command

This command is a wrapper around the **gnuplot** program[58]. **gnuplot** is used to render plots generated by the **tran** and **vtc** commands. The following commands are used to manage those plots:

```
gnuplot                           -- list all active plots

gnuplot <plot> ps [<filename>]    -- print <plot> to ps file <filename>
                                     default filename is <plot-title>.ps.

gnuplot <plot> close              -- close <plot>

gnuplot <plot> display            -- display <plot> in a window.
                                     This is done automatically when
                                     <plot> is created unless
                                     <auto-display>=FALSE

<plot> can be one of:
|  <plot-title>
|  $<plot-number>
|  $!  (last plot)
|  $*  (all plots)
```

Appendix F

Appendix: Rigorous Verification Calculations

I now give detailed I/O for the FenceCalc™ verifications of the two circuit verifications described in Chapter 7.

F.1 Synchronized Pair of Loops

F.1.1 KAST input

```
#global/default parameters
eta        = 0.35;
vDD        = 5;
stat_ratio = 0.05;

#default transconductance and transistor thresholds
kn  = 2;
vTn = 1;
kp  = 1;
vTp = 1;


celem2(m4, n4, o);

# ring m
inv(o, m1);
inv(m1,m2);
inv(m2,m3);
inv(m3,m4);

# ring n
inv(o, n1);
inv(n1,n2);
inv(n2,n3);
inv(n3,n4);
```

F.1.2 KAVL input

```
# load in the circuit,
# initializing transient hypotheses with specified stepsize
load-kast syncloop.kast .0001

# the slewtime isn't as bad if we start measuring it later.
# also we need to increase this for the 'epsilon check'.
epsiset   o- o+   1.6

# we have to adjust the I thresholds of the stages after the C-element
# to deal with its long slewtime and stages after that
# for accurate delay measurement.

epsiset   n1- m1- 1.5    n2- m2- 1.3    n3- m3- 1.1
epsiset   n1+ m1+ 1.3    n2+ m2+ 1.1    n3+ m3+ 1.0

epsilon calc

#calculate all hypotheses
bfs-tol 1.0e-4
bfs-limit 10
bfs dc
bfs tran init

#output some of the hypotheses
tran o+
tran o-
tran o- partial raw
tran m1+
tran m1-
tran m4+
tran m4-
gnuplot $*


#
#check the result
#

#make sure epsilon thresholds are consistent with each other
epsilon check

#verify all hypotheses
bfs-tol 1.0e-6
bfs check

#check transmission-safeness
slew-check

#evaluate final noise tolerance of the proof
#by subtracting all calculation error sources from "eta" for each node.
round-err 1.0e-2  #includes effect of bfs-tol
eta-eff verbose
```

F.1.3 Selected Output

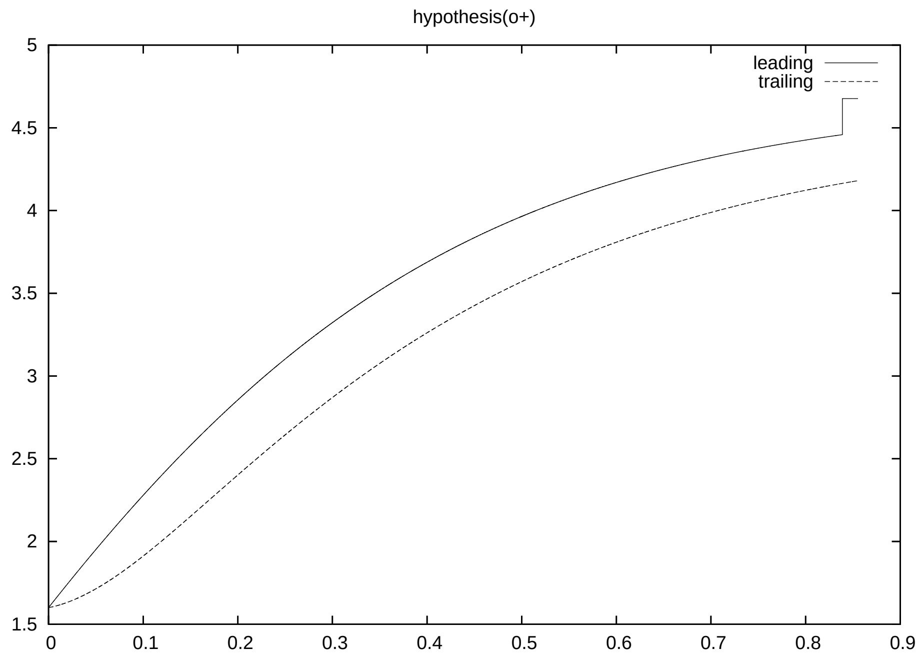


Figure F.1: Complete hypothesis for $o\uparrow$ in the synchronizing loop circuit. The end of the leading bound was padded with a conservative DC value to match the length of the trailing bound.

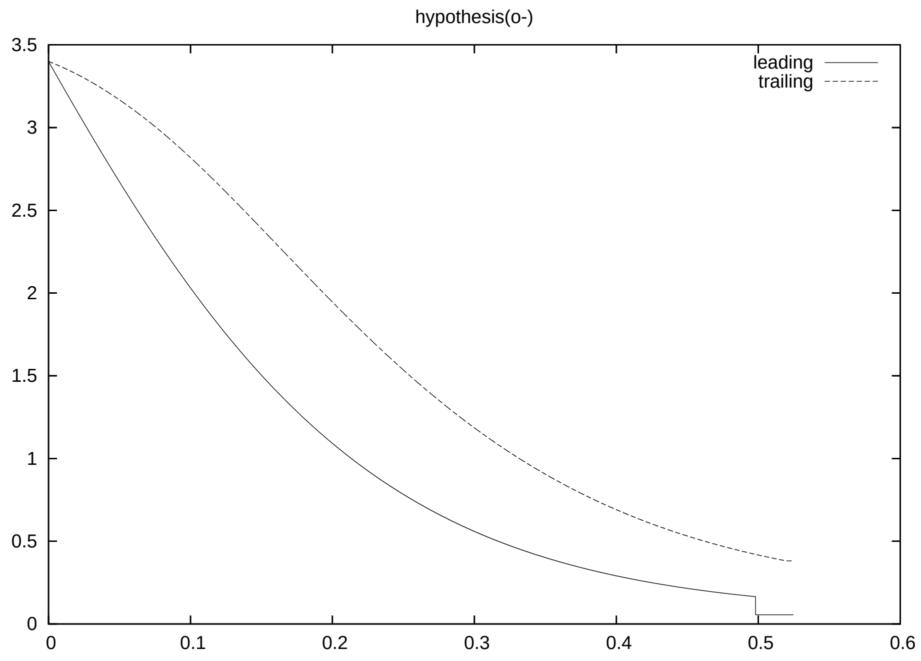


Figure F.2: Complete hypothesis for $o\downarrow$ in the synchronizing loop circuit.

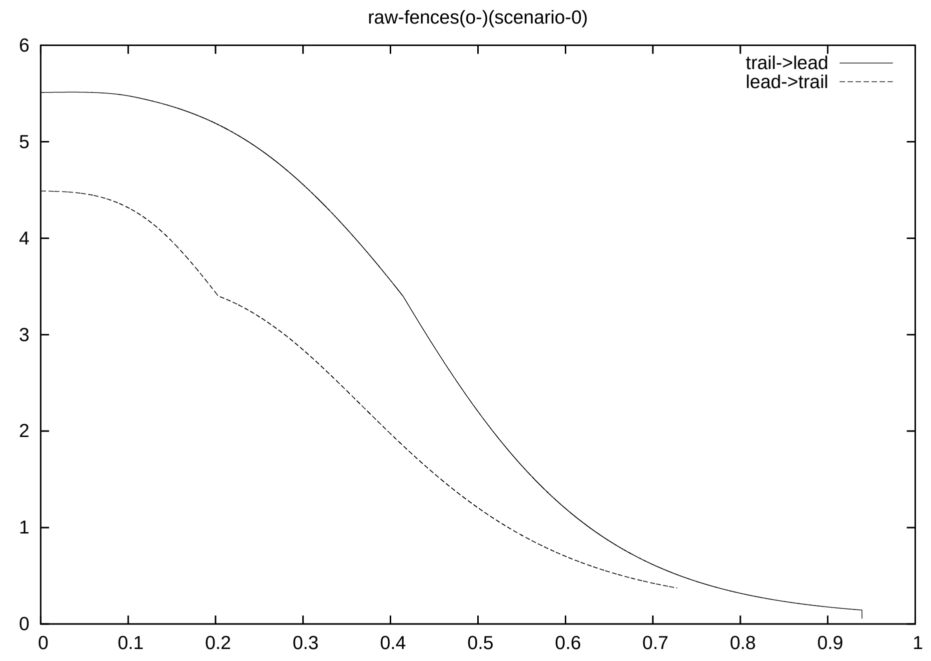


Figure F.3: Partial hypothesis for one of the two $o\downarrow$ scenarios in the synchronizing loop circuit. This includes the calculation of α and α_M, computed in one pass as discussed in Section 6.6.1. This is why there is a kink at $V_{\downarrow I}$.

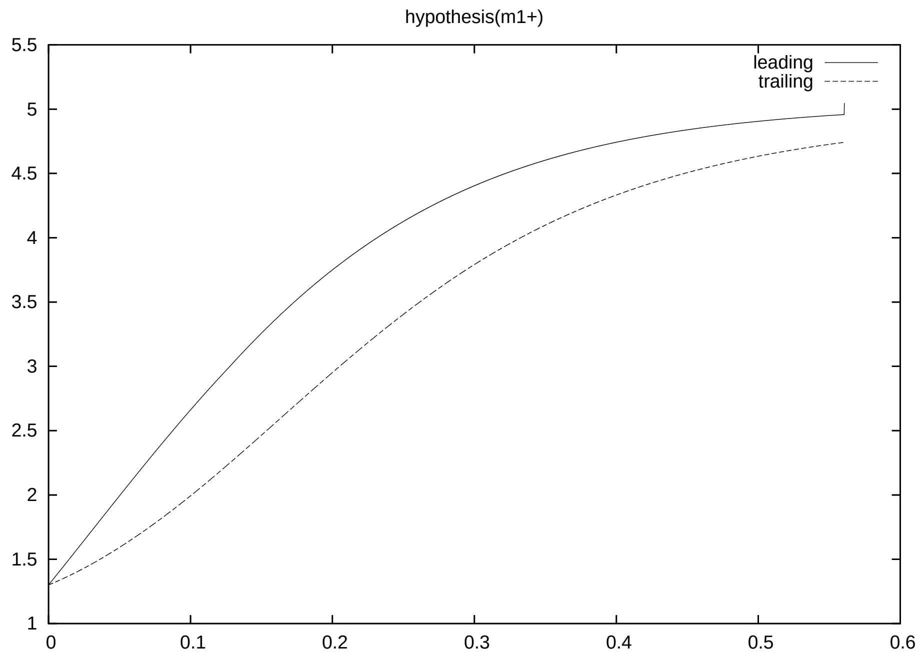


Figure F.4: Complete hypothesis for $m1\uparrow$ in the synchronizing loop circuit.

```
FenceCalc> load-kast syncloop.kast .0001
FenceCalc> epsiset   o- o+   1.6
```

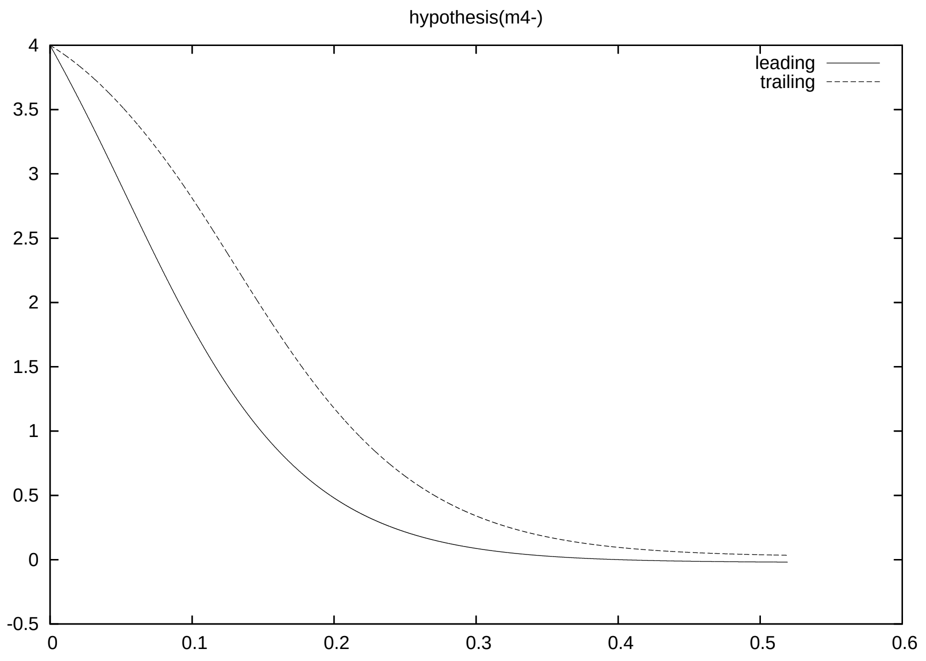


Figure F.5: Complete hypothesis for $m4{\downarrow}$ in the synchronizing loop circuit.

```
FenceCalc> epsiset   n1- m1- 1.5    n2- m2- 1.3    n3- m3- 1.1
FenceCalc> epsiset   n1+ m1+ 1.3    n2+ m2+ 1.1    n3+ m3+ 1.0
FenceCalc> epsilon calc

epsilon n2-[c] calc
n1 in [3.50000,6.50000]v (n1- not initiated)
n2 in [-0.02931,0.06074]v (n2- still within c of rail=0)

epsilon[n2-,c] := 0.0668116

epsilon n2+[c] calc
n1 in [-1.30000,1.30000]v (n1+ not initiated)
n2 in [4.90000,5.05805]v (n2+ still within c of rail=5)

epsilon[n2+,c] := 0.11


:  (skipping)


epsilon o-[c] calc
fighting-staticizer case
m4 in [4.00000,6.00000]v (m4- not initiated)
n4 in [4.00000,6.00000]v (n4- not initiated)
o in [0.04520,0.19821]v (o- still within c of rail=0)

epsilon o-[c] calc (by scenario)
held by staticizer
using default range (not empty range) on node ‘m4’
using default range (not empty range) on node ‘n4’
scenario: input #0 latest (m4- not initiated)
```

```
o in [-0.41588,0.46447]v (o- still within c of rail=0)
scenario: input #1 latest (n4- not initiated)
o in [-0.41588,0.46447]v (o- still within c of rail=0)

epsilon[o-,c] := 0.510913


: (skipping)


FenceCalc> bfs-tol 1.0e-4
FenceCalc> bfs-limit 10
FenceCalc> bfs dc

******************* DC Hypothesis ***************

******* Hypothesis Iteration ********
Input-node phase. Node-graph basis B: n2
Span(cut-set) S: m4 n1 m3 n4 m2 n3 o m1 n2
B - S:

Cycle passes. Transition-graph cutset:
m4- n3+

*** pass 1
worst containment (in pass 1): -1e10

*** pass 2
worst containment (in pass 2): -9.00903e-7

******* Hypothesis Iteration ********
Accumulate enabled; skipping input-node phase.

Cycle passes. Transition-graph cutset:
m4- n3+

*** pass 1
worst containment (in pass 1): -1.76703e-11
FenceCalc> bfs tran init


******************* AC Hypothesis ****************

******* Hypothesis Iteration ********
Input-node phase. Node-graph basis B: n2
Span(cut-set) S: m4 n1 m3 n4 m2 n3 o m1 n2
B - S:

Cycle passes. Transition-graph cutset:
m4- n3+

*** pass 1
update n3+ [acc=FALSE]: containment = -3.99687
update m4- [acc=FALSE]: containment = -3.99367
update n4- [acc=FALSE]: containment = -3.9976
update o+ [acc=FALSE]
calculating fences for scenario: input #0 latest
calculating fences for scenario: input #1 latest
```

```
: containment = -3.39957
update m1- [acc=FALSE]: containment = -3.49881
update n1- [acc=FALSE]: containment = -3.49881
update m2+ [acc=FALSE]: containment = -3.89896
update n2+ [acc=FALSE]: containment = -3.89896
update n3- [acc=FALSE]: containment = -3.89873
update m3- [acc=FALSE]: containment = -3.89873
update n4+ [acc=FALSE]: containment = -3.99876
update m4+ [acc=FALSE]: containment = -3.99876
update o- [acc=FALSE]
calculating fences for scenario: input #0 latest
calculating fences for scenario: input #1 latest
: containment = -3.39927
update m1+ [acc=FALSE]: containment = -3.69905
update n1+ [acc=FALSE]: containment = -3.69905
update m2- [acc=FALSE]: containment = -3.69871
update n2- [acc=FALSE]: containment = -3.69871
update m3+ [acc=FALSE]: containment = -3.99887
worst containment (in pass 1): -3.99887

*** pass 2
update n3+ [acc=FALSE]: containment = -0.577043
update m4- [acc=FALSE]: containment = -1.43216
update n4- [acc=FALSE]: containment = -0.464585
update o+ [acc=FALSE]
calculating fences for scenario: input #0 latest
calculating fences for scenario: input #1 latest
: containment = -0.0748316
update m1- [acc=FALSE]: containment = -0.0956479
update n1- [acc=FALSE]: containment = -0.0956479
update m2+ [acc=FALSE]: containment = -0.0824946
update n2+ [acc=FALSE]: containment = -0.0824946
update n3- [acc=FALSE]: containment = -0.0974282
update m3- [acc=FALSE]: containment = -0.0974282
update n4+ [acc=FALSE]: containment = -0.0525842
update m4+ [acc=FALSE]: containment = -0.0525842
update o- [acc=FALSE]
calculating fences for scenario: input #0 latest
calculating fences for scenario: input #1 latest
: containment = -0.0292191
update m1+ [acc=FALSE]: containment = -0.0199324
update n1+ [acc=FALSE]: containment = -0.0199324
update m2- [acc=FALSE]: containment = -0.0240267
update n2- [acc=FALSE]: containment = -0.0240267
update m3+ [acc=FALSE]: containment = -0.0142127
worst containment (in pass 2): -1.43216


: (skipping)


*** pass 5
update n3+ [acc=FALSE]: containment = -0.0000211963
update m4- [acc=FALSE]: containment = -0.0000209979
update n4- [acc=FALSE]: containment = -0.0000209979
update o+ [acc=FALSE]
calculating fences for scenario: input #0 latest
calculating fences for scenario: input #1 latest
```

```
: containment = -5.28563e-6
update m1- [acc=FALSE]: containment = -6.90453e-6
update n1- [acc=FALSE]: containment = -6.90453e-6
update m2+ [acc=FALSE]: containment = -4.91001e-6
update n2+ [acc=FALSE]: containment = -4.91001e-6
update n3- [acc=FALSE]: containment = -5.48595e-6
update m3- [acc=FALSE]: containment = -5.48595e-6
update n4+ [acc=FALSE]: containment = -2.50509e-6
update m4+ [acc=FALSE]: containment = -2.50509e-6
update o- [acc=FALSE]
calculating fences for scenario: input #0 latest
calculating fences for scenario: input #1 latest
: containment = -1.30589e-6
update m1+ [acc=FALSE]: containment = -8.81998e-7
update n1+ [acc=FALSE]: containment = -8.81998e-7
update m2- [acc=FALSE]: containment = -9.94972e-7
update n2- [acc=FALSE]: containment = -9.94972e-7
update m3+ [acc=FALSE]: containment = -4.88793e-7
worst containment (in pass 5): -0.0000211963
FenceCalc>


: (skipping)

FenceCalc> bfs-tol 1.0e-6
FenceCalc> bfs check


******************* DC Hypothesis ***************

final containment = -0


******************* AC Hypothesis ****************

final containment = -4.88793e-7

FenceCalc> slew-check
mindelay([n2+] n3- n4+ o- n1+ n2-) - maxslew(n3-) = 1.0164s - 0.4839s
mindelay([n1+] n2- n3+ n4- o+ n1-) - maxslew(n2-) = 1.0308s - 0.436s
mindelay([m4+] o- n1+ n2- n3+ n4-) - maxslew(o-) = 1.0053s - 0.5249s
mindelay([m4-] o+ m1- m2+ m3- m4+) - maxslew(o+) = 1.0449s - 0.8555s
mindelay([n4+] o- m1+ m2- m3+ m4-) - maxslew(o-) = 1.0053s - 0.5249s
mindelay([m3-] m4+ o- m1+ m2- m3+) - maxslew(m4+) = 0.9958s - 0.812s
mindelay([n4-] o+ n1- n2+ n3- n4+) - maxslew(o+) = 1.0449s - 0.8555s
mindelay([m2-] m3+ m4- o+ m1- m2+) - maxslew(m3+) = 1.0338s - 0.783s
mindelay([n3-] n4+ o- n1+ n2- n3+) - maxslew(n4+) = 0.9958s - 0.812s
mindelay([m1-] m2+ m3- m4+ o- m1+) - maxslew(m2+) = 1.0194s - 0.6827s
mindelay([o-] m1+ m2- m3+ m4- o+) - maxslew(n1+ m1+) = 0.9719s - 0.5607s
mindelay([n2-] n3+ n4- o+ n1- n2+) - maxslew(n3+) = 1.0338s - 0.783s
mindelay([n1-] n2+ n3- n4+ o- n1+) - maxslew(n2+) = 1.0194s - 0.6827s
mindelay([m4+] o- m1+ m2- m3+ m4-) - maxslew(o-) = 1.0053s - 0.5249s
mindelay([m4-] o+ n1- n2+ n3- n4+) - maxslew(o+) = 1.0449s - 0.8555s
mindelay([m3+] m4- o+ m1- m2+ m3-) - maxslew(m4-) = 1.0544s - 0.5192s
mindelay([n4+] o- n1+ n2- n3+ n4-) - maxslew(o-) = 1.0053s - 0.5249s
mindelay([n4-] o+ m1- m2+ m3- m4+) - maxslew(o+) = 1.0449s - 0.8555s
mindelay([m2+] m3- m4+ o- m1+ m2-) - maxslew(m3-) = 1.0164s - 0.4839s
mindelay([n3+] n4- o+ n1- n2+ n3-) - maxslew(n4-) = 1.0544s - 0.5192s
mindelay([m1+] m2- m3+ m4- o+ m1-) - maxslew(m2-) = 1.0308s - 0.436s
mindelay([o+] m1- m2+ m3- m4+ o-) - maxslew(n1- m1-) = 1.0783s - 0.5311s

worst path: [m3-] m4+ o- m1+ m2- m3+
FenceCalc> round-err 1.0e-2  #includes effect of bfs-tol
FenceCalc> eta-eff verbose
node      eff =     eta - round -(     red=     h(     P+     K*     M))
n2    0.25908 = 0.35000 -  0.01 -( 0.08092=0.0001( 1.455+ 24.52* 32.94))
m1    0.20966 = 0.35000 -  0.01 -( 0.13034=0.0001( 4.016+ 28.25*    46))
o     0.29132 = 0.35000 -  0.01 -( 0.04868=0.0001(0.5759+ 28.91* 16.82))
n3    0.25918 = 0.35000 -  0.01 -( 0.08082=0.0001(0.5128+ 24.52* 32.94))
m2    0.25908 = 0.35000 -  0.01 -( 0.08092=0.0001( 1.455+ 24.52* 32.94))
n4    0.25920 = 0.35000 -  0.01 -( 0.08080=0.0001(0.2406+ 24.52* 32.94))
m3    0.25918 = 0.35000 -  0.01 -( 0.08082=0.0001(0.5128+ 24.52* 32.94))
n1    0.20966 = 0.35000 -  0.01 -( 0.13034=0.0001( 4.016+ 28.25*    46))
m4    0.25920 = 0.35000 -  0.01 -( 0.08080=0.0001(0.2406+ 24.52* 32.94))
          v/s =     v/s -   v/s -(     v/s=     s( v/s^2+   1/s*   v/s))
FenceCalc> quit # success if we got this far
```

F.2 Chain of Dual-Rail WCHB Buffers

F.2.1 KAST input

```
#global/default parameters
eta        = 0.09;
vDD        = 5;
stat_ratio = 0.05;

#default transconductance and transistor thresholds
kn  = 2;
vTn = 1;
kp  = 1;
vTp = 1;


#first buffer

celem2(l0,re,_r0);
celem2(l1,re,_r1);

inv(_r0, r0);
inv(_r1, r1);

nand2(_r0,_r1,___le);
inv(___le,__le);
inv(__le,_le);
inv(_le,le);


#second buffer

celem2(r0,se,_s0);
celem2(r1,se,_s1);

inv(_s0, s0);
inv(_s1, s1);

nand2(_s0,_s1,___re);
inv(___re,__re);
inv(__re,_re);
inv(_re,re);


#environment
#         cause/eff cause/eff
handshake le+ l0+ le- l0-
handshake le+ l1+ le- l1-
handshake s0+ se- s0- se+
handshake s1+ se- s1- se+
```

F.2.2 KAVL input

As with the synchronizing loop circuit, we must specify the I/C thresholds that define the observation rule. In addition, however, the WCHB chain has an environment, so we must specify the hypotheses on the environment's outputs using the `tran` command.

```
# load in the circuit,
# initializing transient hypotheses with specified stepsize
load-kast bufchain.kast .0003


#set hypotheses for input nodes
epsilon se+ i 1.0
epsilon se+ c 0.5

epsilon se- i 1.0
epsilon se- c 0.5

tran se- .05 30
tran se+ .05 30


epsilon l0+ i 1.0
epsilon l0+ c 0.5

epsilon l0- i 1.0
epsilon l0- c 0.5

tran l0- .05 30
tran l0+ .05 30



epsilon l1+ i 1.0
epsilon l1+ c 0.5

epsilon l1- i 1.0
epsilon l1- c 0.5
```

```
tran l1- .05 30
tran l1+ .05 30


#adjust I thresholds of internal nodes

#celems
epsiset  _s0+ _s1+ _s0- _s1- 1.8
epsiset  _r0+ _r1+ _r0- _r1- 1.6

epsiset __le+ __re+ 1.6
epsiset __le- __re- 2.0

epsiset _le+ _re+ 1.6
epsiset _le- _re- 2.0

#nand2s
epsiset  ___le+ ___le- 1.6
epsiset  ___re+ ___re- 1.6

#1-after celem
epsiset  r0+ r1+ s0+ s1+ 1.4
epsiset  r0- r1- s0- s1- 1.8

#1-after nand2
epsiset __le+ __le- __re+ __re- 1.8



bfs dc

epsilon calc

#calculate all hypotheses
bfs-tol 5.0e-5
bfs-limit 13
bfs tran init



#
#check the result
#

#make sure epsilons are consistent with each other
epsilon check

#verify all hypotheses
bfs-tol 1.1e-5
bfs check

#check transmission-safeness
slew-check

handshakes

eta-eff verbose
eta-eff verbose 4
```

F.2.3 Selected Output

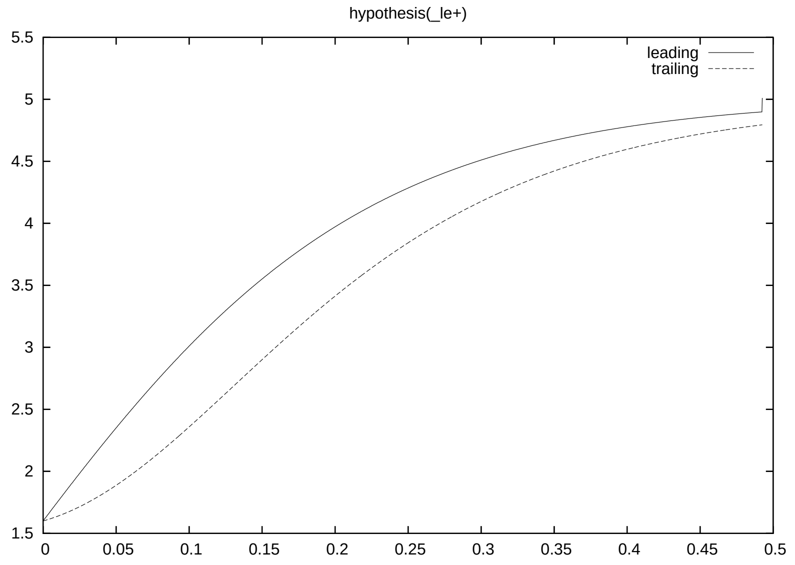


Figure F.6: Complete hypothesis for $\_le\uparrow$ in the WCHB buffer chain.

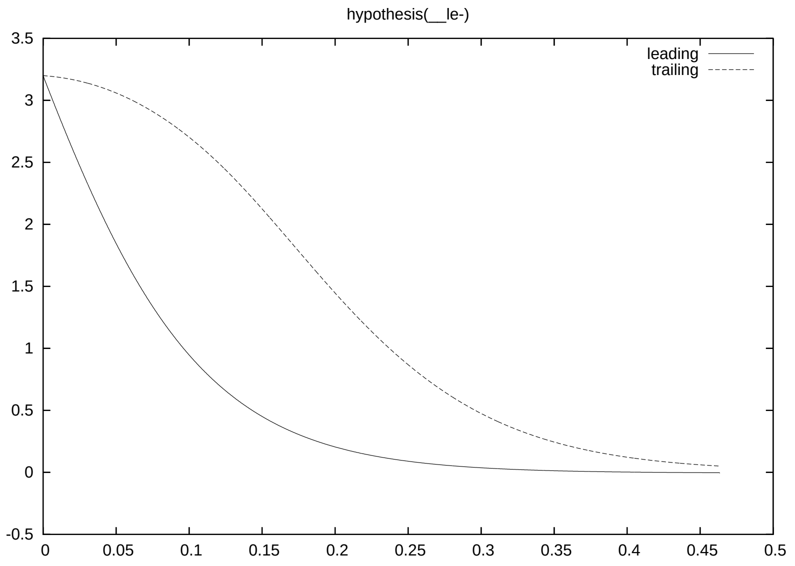


Figure F.7: Complete hypothesis for $\_\_le\downarrow$ in the WCHB buffer chain.

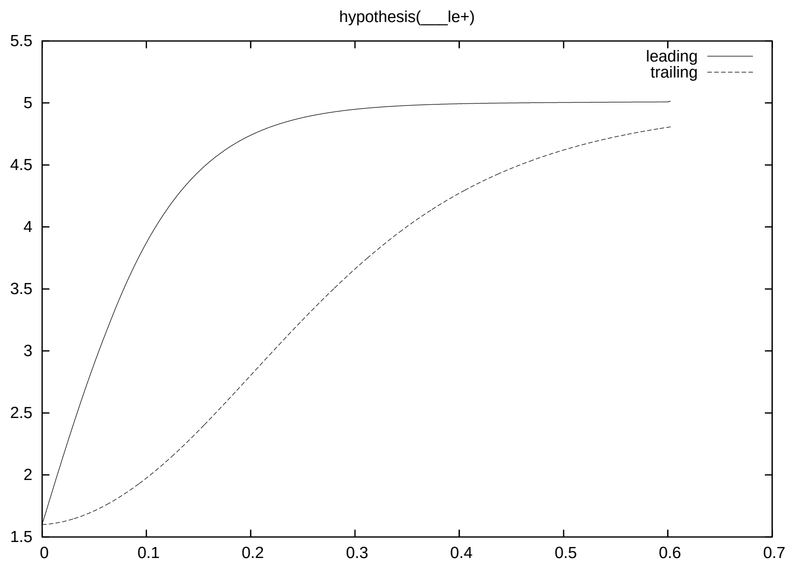


Figure F.8: Complete hypothesis for $\_\_\_le\uparrow$ in the WCHB buffer chain.

Bibliography

[1] Alain J. Martin, "Compiling Communicating Processes into Delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, no. 4, 1986.

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

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

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

[5] Takashi Nanya and Yoichiro Ueno and Hiroto Kagotani and Masashi Kuwako and Akihiro Takamura, "TITAC: Design of a Quasi-Delay-Insensitive Microprocessor," *IEEE Design and Test of Computers*, vol. 11, no. 2, pp. 50–63, 1994.

[6] *Merriam-Webster's Collegiate Dictionary*, Merriam-Webster, Incorporated, 10th edition, 2001.

[7] Karl Papadantonakis, "Design Rules for Non-Atomic Implementations of PRS," Technical Report caltechCSTR/2005.001, Caltech, Pasadena, CA, 2005.

[8] John Hamal Hubbard and Beverly H. West, *Differential Equations: A Dynamical Systems Approach*, Springer, 1991.

[9] N. Francez, *Fairness*, Springer-Verlag, New York, 1986.

[10] K. Mani Chandy, *Parallel Program Design: A Foundation*, Addison-Wesley, Reading, Massachusetts, 1988.

[11] Stephen A. Ward and Robert H. Halstead, *Computation Structures*, McGraw-Hill, 1990.

[12] Keith Hanna, "Reasoning about Real Circuits," in *Proc 7th Intnl Conf on Higher-Order Logic, Theorem Proving and its Applications. Malta*. September 1994, Springer Verlag.

[13] David E. Muller, "Asynchronous Logic and Applications to Information Processing," *Switching Theory and Space Technology*, 1963.

[14] Jin-fuw Lee and Donald T. Tang, "An Algorithm for Incremental Timing Analysis," in *Proc. 32nd ACM/IEEE Conference on Design Automation*, San Francisco, 1995.

[15] D. B. Armstrong and A.D. Friedman and P.R. Menon, "Design of Asynchronous Circuits Assuming Unbounded Gate Delays," *IEEE Transactions on Computers*, vol. C-18, no. 12, pp. 1110–1120, December 1969.

[16] Robin Milner, "A Calculus of Communicating Systems," *Lecture Notes in Computer Science*, , no. 92, 1980.

[17] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, London, U.K., 1984.

[18] K. Mani Chandy, *An Introduction to Parallel Programming*, Jones and Bartlett, Boston, Massachusetts, 1992.

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

[20] Luciano Lavagno and Kurt Keutzer and Alberto Sangiovanni-Vincentelli, "Synthesis of Hazard-Free Asynchronous Circuits with Bounded Wire Delays," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 1, pp. 61–86, January 1995.

[21] Rajit Manohar and Alain J. Martin, "QDI circuits are Turing-Complete," *Second International Symposium on Advanced Research in Asynchronous Circuits and Systems*, March 1996.

[22] Andrew Matthew Lines, "Pipelined Asynchronous Circuits," M.S. thesis, Caltech, 1995.

[23] Kees van Berkel, "Beware the Isochronic Fork," *Integration, the VLSI Journal*, vol. 13, no. 2, pp. 103–128, June 1992.

[24] P. Beerel and T. H-Y. Meng., "Automatic Gate-Level Synthesis of Speed-Independent Circuits," in *Proc. International Conf. Computer-Aided Design (ICCAD).* November 1993, pp. 581–587, IEEE Computer Society Press.

[25] Tam-Anh Chu, "On the Models for Designing VLSI Asynchronous Digital Circuits," *Integration, the VLSI Journal*, vol. 4, no. 2, pp. 99–113, June 1986.

[26] A. Kondratyev and M. Kishinevsky and B. Lin and P. Vanbekbergen and A. Yakovlev, "Basic Gate Implementation of Speed-Independent Circuits," in *Proceedings of the Design Automation Conference*, June 1994, pp. 56–62.

[27] R. Alur, *Techniques for Automatic Verification of Real-Time Systems*, Ph.D. thesis, Stanford University, 1991.

[28] David L. Dill, "Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits," *ACM Distinguished Dissertations*, 1989.

[29] P. Vanbekbergen and A. Wand and K. Keutzer, "A Design and Validation System for Asynchronous Circuits," *Proc. ACM/IEEE Design Automation Conference*, June 1995.

[30] Oriol Roig and Jordi Cortadella and Enric Pastor, "Hierarchical Gate-Level Verification of Speed-Independent Circuits," *Asynchronous Design Methodologies*, pp. 129–137, May 1995.

[31] A. Kondratyev and M. Kishinevsky and B. Lin and P. Vanbekbergen and A. Yakovlev, "On the Conditions for Gate-Level Speed-Independence of Asynchronous Circuits," in *Proceedings of the ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 1993.

[32] Karl Papadantonakis, "Stable PRS are Deterministic," Technical Report caltechCSTR/2003.003, Caltech, 2003.

[33] Karl Papadantonakis, "What is Deterministic CHP, and is Slack Elasticity that Useful?," M.S. thesis, Caltech, 2002.

[34] Norbert R. Malik, *Electronic Circuits*, Prentice Hall, 1995.

[35] W. Nagel, "SPICE 2 – A Computer Program to Simulate Semiconductor Circuits," EECS Memo M520, UC Berkeley, 1975.

[36] I. L. Wemple and A. T. Yang, "Mixed-Signal Switching Noise Analysis Using Voronoi-Tessellated Substrate Macromodels," in *Proceedings of the Design Automation Conference*, 1995.

[37] Alain J. Martin, "Tomorrow's Digital Hardware will be Asynchronous and Verified," *Information Processing*, vol. 1, pp. 684–695, 1992.

[38] A. Degloria and P. Faraboschi and 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.

[39] Mika Nystroem, "alint," software, 1997.

[40] Mark R. Greenstreet, "Verifying Safety Properties of Differential Equations," in *Proceedings of the 1996 Conference on Computer Aided Verification*, New Brunswick, NJ, July 1996, pp. 277–287.

[41] Mark R. Greenstreet and Ian Mitchell, "Reachability Analysis Using Polygonal Projections," Tech. Rep. www.cs.ubc.ca/ mrg/mypapers/hs99.ps, University of British Columbia, 1999.

[42] Mika Nystroem, *Asynchronous Pulse Logic*, Ph.D. thesis, Caltech, May 2001.

[43] Ivan E. Sutherland, "Logical Effort: Designing for Speed on the Back of an Envelope," *IEEE Advanced Research in VLSI*, pp. 1–16, 1991.

[44] Tak Kwan Lee, *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits*, Ph.D. thesis, Caltech, May 1995.

[45] S. M. Burns, "Performance Analysis and Optimization of Asynchronous Circuits," Tech. Rep. Caltech-CS-TR-91-01, Caltech, December 1990.

[46] Jawahar Jain and Rajarshi Mukherjee and Masahiro Fujita, "Advanced Verification Techniques Based on Learning," in *Proc. 32nd ACM/IEEE Conference on Design Automation*, San Francisco, 1995, pp. 420–426.

[47] Jens Spars/o and J/orgen Staunstrup, "Delay-Insensitive Multi-Ring Structures," *Integration, the VLSI Journal*, vol. 15, no. 3, pp. 313–340, 1993.

[48] T. E. Williams, "Performance of Iterative Computation in Self-Timed Rings," *Journal of VLSI Signal Processing*, vol. 7, no. 1-2, pp. 17–32, 1994.

[49] Alain J. Martin and P. J. Hazewindus, "Testing Delay-Insensitive Circuits," in *Proceedings of the Conference on Advanced Research in VLSI (ARVLSI)*, March 1991.

[50] Wonjin Jang and Alain J. Martin, "SEU-tolerant QDI Circuits," in *Proc. 11th IEEE International Symposium on Asynchronous Systems and Circuits (ASYNC)*, March 2005.

[51] Neil H. E. Weste and David Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Addison-Wesley, third edition, 2005.

[52] R. Cori and D. Lascar, *Mathematical Logic: A Course with Exercises*, Oxford, 1993.

[53] Paul Penzes, *Energy-delay Complexity of Asynchronous Circuits*, Ph.D. thesis, Caltech, May 2002.

[54] Jan M. Rabaey, *Digital Integrated Circuits: A Design Perspective*, Electronics and VLSI. Prentice Hall, 1996.

[55] Rajit Manohar, "Caltech Asynchronous Synthesis Tools (CAST)," software, 1997.

[56] R. Anglada and A. Rubio, "An approach to crosstalk effect analyses and avoidance techniques in digital CMOS VLSI circuits," *International Journal of Electronics*, vol. 6, no. 5, pp. 9–17, 1988.

[57] John Hamal Hubbard and Barbara Burke Hubbard Hubbard, *Vector Calculus, Linear Algebra, and Differential Forms: A Unified Approach*, Prentice Hall, 1999.

[58] Thomas Williams and Colin Kelley, "gnuplot," software, http://www.gnuplot.info/, 1986.

Index