PROVING CORRECTNESS OF ASYNCHRONOUS CIRCUITS USING TEMPORAL LOGIC

Mark Joseph Bennett

April 1986
CSD-860089

UNIVERSITY OF CALIFORNIA

Los Angeles

Proving Correctness of Asynchronous Circuits Using Temporal Logic

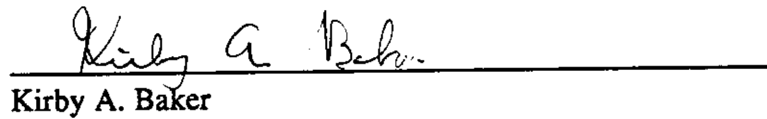
A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Computer Science

by

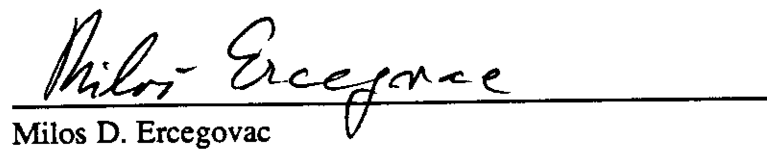
Mark Joseph Bennett

1986

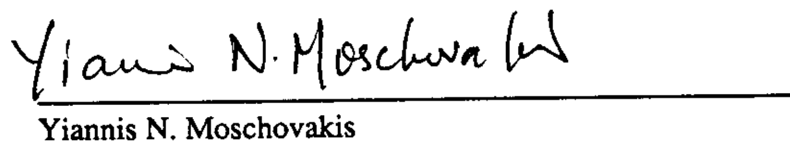
The dissertation of Mark Joseph Bennett is approved.



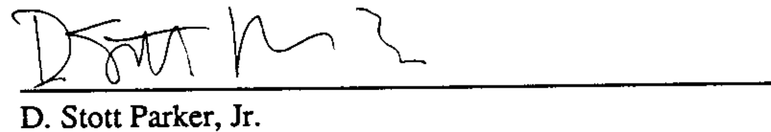
Kirby A. Baker



Milos D. Ercegovac



Yiannis N. Moschovakis



D. Stott Parker, Jr.

David F. Martin, Committee Chair

University of California, Los Angeles
1986

To my parents,

whose constant love and support

made this possible

TABLE OF CONTENTS

Page

1. Introduction 1
 1.1 Overview....1
 1.2 Asynchronous Circuits 3
 1.3 Hierarchical Verification....4
 1.4 Safety and Liveness....5
 1.5 Theorem Prover....5
 1.6 Contributions of this Thesis....6
 1.7 Organization of this Thesis....7
2. Temporal Logic 8
 2.1 Definition of Temporal Logic 8
 2.2 Axiomatization of Temporal Logic....12
 2.3 PTL Timing Diagrams....18
 2.4 PTL Theorems....19
 2.5 Describing Hardware with Temporal Logic 25
3. Related Work....31
 3.1 Semantics-Based Verification....31
 3.1.1 Milner's CCS 32
 3.1.2 Gordon's Synchronous Hardware Model....37
 3.1.3 Milne's CIRCAL....41
 3.2 Predicate Logic-Based Verification....42
 3.2.1 Wagner's Hardware Verification System 43
 3.2.2 Patterson's STRUM....43
 3.3 Temporal Logic-Based Verification 44
 3.3.1 Malachi and Owicki's Temporal Specifications 44
 3.3.2 Moszkowski's ITL 45
 3.3.3 Bochmann's Arbiter Proof....47
 3.3.4 Model Check Approach 50
 3.4 Relation to Our Work 51
4. Steady-State Properties 53
 4.1 A Proof System 54
 4.1.1 Steady-State Behavior....54
 4.1.2 Gate Properties 55
 4.1.3 Input Elimination Rules 62
 4.1.4 Stability Implication Rules 65
 4.1.5 Inverter-Inverter Stability Rules 65
 4.1.6 Inverter-Gate Stability 66
 4.1.7 Coincidence Rule....69
 4.1.8 Additional Theorems 70
 4.2 Proving Safety and Liveness....72
 4.2.1 Latch Properties....72
 4.2.2 Correctness of an Asynchronous Controller....78
5. Global-Time Properties....89
 5.1 Proof Tools....90
 5.1.1 Forward and Backward Reasoning....90
 5.1.2 Transition Safety Properties....99
 5.1.3 Proof Rules....100
 5.2 Safety Strategy 101

5.2.1 Latch Safety........102
5.2.2 Controller Safety........102
5.3 Liveness Strategy........106
5.3.1 Latch Liveness........106
5.3.2 Controller Liveness........109
5.4 Discussion........119
6. Execution Graph Method........121
6.1 Execution Graphs........121
6.2 Checking Specifications........129
6.2.1 Checking Safety Specifications........132
6.2.2 Checking Liveness Specifications........136
6.2.3 An Example........140
6.3 Generation of SLEGs........143
6.4 Application to Self-Timed Circuits........147
6.4.1 Self-Timed Protocol........147
6.4.2 Module Axioms........148
6.4.3 State Template and SLEG........151
6.4.4 Checking PL Specifications........152
7. Anomalies........157
7.1 Proving the Presence of Critical Races........157
7.1.1 Grammar Operators........158
7.1.2 Reasoning About Races........160
7.2 Stability and Instability........165
7.3 Proving the Absence of Critical Races........168
8. Conclusions and Suggestions for Future Research........171
8.1 Summary........171
8.2 Relation to Other Work........174
8.3 Future Directions........176
8.4 Conclusions........176
Appendix A. Proofs of Theorems 4.1 - 4.7........178

ACKNOWLEDGMENTS

First of all, I would like to express my appreciation to Professor David Martin, who directed this research. His thoughtfulness, patience, understanding, and insight were invaluable during this endeavor. He has been a mentor, motivator, source of ideas, and friend through my graduate studies.

I am grateful to my committee members, Professors Milos Ercegovac, Stott Parker, Kirby Baker, and Yiannis Moschovakis, for serving on my doctoral committee. Extra thanks go to Professor Milos Ercegovac for motivating the ideas of Chapter 7.

The people at System Development Corporation, Bob MacGregor, Margaret Reid-Miller, Margie Templeton, Iris Kameny, Beatrice Oshika, and Harvey Gold, have been very understanding and supportive during my doctoral studies. They always sympathized with my school commitments and were willing to be flexible about work responsibilities and schedules.

Ron Krupp, formerly at Argonne and now at Bell Laboratories, was kind enough to hire me during my undergraduate summers when I needed both practical experience and cash. Bill Niedfeldt, Pat Baldwin, and Phil Ridinger, were very supportive of my career goals and helped me meet them by giving me excellent experience at Bell Laboratories.

On the more personal side, I would like to thank my friends Erich, Karen, Kevin, Marc, and Scott for making "doing" L.A. a memorable experience. I am also indebted to Erich for inspiring me with many stimulating discussions about hardware. Thanks go to Kevin, for getting me drunk often enough to allow me to keep my sanity through it all.

Vacations back home in Chicago were always exciting and fun thanks to my friends Mike, Annette, Tom, Carol, Dean, Ed, and Einar.

My friend Steve always lifted my spirits over the years, through thick and thin, encouraging me to keep plowing forward through school.

My brothers and sisters, Patti, Tom, Joanne, Bob and Jim, gave me their invaluable love and reminded me that it would all be worthwhile. My brother-in-law Rich helped me see early on the intellectual rewards of science and engineering through lengthy sessions of chewing the fat.

A special thank you goes to Robin, who recently came into my life and touched me in a way only she can.

Finally, I would like to begin to express my graditude to my dear parents, Pat and Ben. From the start, they instilled in me the values and qualities necessary to undertake such a monumental endeavor. Their insistance on hard work and disipline combined with their unyielding love was the perfect formula for achieving high goals. Without their kindness, understanding, support, generousity, and love I never could have achieved this.

VITA

September 19, 1959	Born, Chicago, Illinois
1977-1979	Programmer Argonne National Laboratory
1980	Technical Associate Bell Laboratories Naperville, Illinois
1981	Teaching Assistant, University of Iowa
1981	The Honors Certificate of Achievement University of Iowa
1981	B.S., (Cum Laude) University of Iowa
1981-1982	Teaching Assistant, University of Southern California
1982	M.S., University of Southern California
1982-1983	Member of Technical Staff Bell Laboratories Naperville, Illinois
1982-1983	School of Engineering Fellowship University of Califonia, Los Angeles
1983	Teaching Associate, University of Califonia, Los Angeles
1983-1986	Research Scientist System Development Corporation Santa Monica, California

ABSTRACT OF THE DISSERTATION

Proving Correctness of Asynchronous Circuits
Using Temporal Logic

by

Mark Joseph Bennett
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 1986
Professor David F. Martin, Committee Chair

The recent advances made in VLSI have prompted new ideas for hardware. As it becomes more difficult to rely upon a global clock and as the need for increased speed arises, asynchronous circuits become more attractive. Since asynchronous circuits operate at a speed determined by the input and gate delays only, they can operate faster than their synchronous counterparts and generate completion signals. Unfortunately current validation techniques are cumbersome and do not instill much confidence in designs.

Proving correctness in a deductive system for propositional temporal logic (PTL) is shown here to be an attractive validation alternative. Axioms describing the behavior of atomic elements (gates or modules) are formulated in PTL. Desired properties of the circuit are formulated as specifications in PTL. As gates or modules are composed to form larger modules, their corresponding PTL formulas are combined

using PTL theorems. Using the methods introduced here, the specifications can be proved correct with respect to the axioms. Verified examples range from latches to a self-timed asynchronous pipeline.

A method for proving steady-state circuit properties is introduced which is based upon a deductive system for temporal logic. A method for proving global-time properties is also presented which is an extension of the steady-state method. Proving global-time properties involves a collection of heuristics such as forward reasoning for liveness, backward reasoning for safety, and proving transition safety and non-interference properties. PTL timing diagrams are introduced as a tool for informally reasoning about complex until-formulas.

A graph representation of a circuit's execution is provided with the Execution Graph Method (EGM), which is introduced to address the lack of behavioral information in a circuit schematic. The execution graph is formed from a set of PTL formulas proved from the axioms. Safety and liveness properties can be shown to hold on the execution graph by using an algorithm which traverses paths in a depth-first manner. It is shown that EGM can be used to verify gate- or module-level circuits hierarchically.

CHAPTER 1

Introduction

1.1 Overview

The recent advances made in VLSI device technology, VLSI circuit design, and VLSI CAD systems have made possible the rapid design of computer hardware. No longer do engineers rely on a few standard logic chips. The age of more custom and more complex systems is here. As a result, there are more types of computer hardware than ever before.

The intent of this work is to increase the level of confidence in hardware designs by constructing mathematical proofs of correctness. Verifying hardware through correctness proofs is a way of convincing oneself that no errors exist in the design. The formalism used for describing and writing assertions about the hardware will be temporal logic. Temporal logic is a logic which incorporates the notion of time into the traditional formal reasoning system of first-order logic, and as such it is particularly useful when timing is the main issue. Temporal logic, in fact, is a formalization of the timing diagrams used by designers.

Hardware designers currently use various validation approaches at different

stages of the design process to increase the level of confidence in the design:

- logic simulation
- analog simulation
- testing

Logic simulation is an execution of the circuit with logic values. The circuit is essentially executed with certain input values and initial storage values. The approach cannot account for every combination of input and storage values (called a *case*) except in simple circuits since the number of possible input values is of exponential order.

Analog simulation consists of modelling the circuit as a network of analog devices. Gate-level properties such as switching time can be determined, and it is here that the effects of parasitic capacitances and wire resistance are studied. This step is necessary to implement the circuit in a particular device technology, such as nMOS or CMOS.

Testing is done after a prototype is available. In the case of VLSI, it has the disadvantage that the chip must first be fabricated, which is very costly. Testing also cannot account for every possible case, although automatic test equipment is making this less true.

Verification should be done before any of the other processes are done. Verification is not a substitute for any of them, but it is one more way to increase the level of confidence. It has the property of being able to handle every case, since it consists of mathematical proofs.

While verification might not be warranted by conventional applications, it is imperative in security applications where true confidence requires a proof that the

system is logically secure, and in life-critical applications where anything less than absolute confidence is unacceptable.

Hardware may be roughly classified as being either synchronous or asynchronous. Synchronous systems have a common clock throughout. Most register transfer systems, ALUs, and memories may be considered synchronous. Asynchronous systems may either have no clock or just appear to have no clock to the external observer. Arbiters, peripheral controllers, and self-timed VLSI circuits [Mea80] are examples of asynchronous systems. As the trend toward architectural support for operating system functions continues [Dit82] more circuits which can be viewed in an asynchronous framework will be embedded in hardware. Examples of these include hardware priority queues [Lei81] and FIFO buffers [Mea80], items previously implemented in software.

For verification purposes, synchronous systems are analogous to sequential programs. The primary properties of interest are partial correctness and termination. Asynchronous circuits possess properties like those of concurrent programs such as mutual exclusion (in the case of circuits such as the arbiter [Sei80b, Boc82]) and responsiveness. The ultimate goal of hardware verification is to treat both synchronous and asynchronous systems. Although synchronous systems can be treated with traditional first-order logic as demonstrated by Wagner [Wag77], we believe that having one specification language, temporal logic, capable of dealing with *both* function and timing will prove beneficial.

1.2 Asynchronous Circuits

Asynchronous communication using request/acknowledge signalling is a useful technique when synchronizing two devices with different clocks. As VLSI and WSI

(wafer-scale integration) clock distribution becomes difficult due to the excessive delays in long wires, asynchronous systems and globally-asynchronous locally-synchronous systems [Cha84] become attractive. In addition, asynchronous systems are generally faster. Synchronous systems must be designed using a worst-case assumption. All devices are reduced to the speed of the slowest device. But using existing validation methods for asynchronous circuits, including race and hazard analysis [Fri75] and those mentioned in Section 1.1, are not exhaustive and thus are prone to ignore faults.

In this thesis, we concentrate on *speed-independent* [Arm69, Ung69] or *Muller* [Mul59, Mil65] asynchronous circuits. Speed-independent circuits operate correctly regardless of the gate delays. Some form of completion signalling is necessary, and adjacent modules communicate through this completion or request/acknowledge signalling. Those circuits using request/acknowledge signalling are called *self-timed.*

Through most of the thesis gates are assumed to have unbounded, but finite, delay. Lines are assumed to have no delay. This simplifies the analysis without loss of generality since line delays can be included in gate delays or modelled as non-inverting delay elements.

1.3 Hierarchical Verification

In order to manage the complexity of the verification process and limit the number of circuit states which need to be considered at one time, verification is done hierarchically, beginning with gates as the lowest-level devices and working up to sequential machines. Initial assumptions (called *axioms*) describing the properties of the lowest-level devices are stated and then the specifications describing the properties of larger modules are proved correct relative to the axioms. The specifications of the

larger modules are then taken as assumptions, and specifications of even larger modules composed of these are proved correct. The process continues in a bottom-up fashion. When complete correctness has been proven, then it has been accomplished with only the lowest-level device axioms assumed.

The *functional specification* technique is employed where the specifications of more complex modules are functions of the specifications of the simpler constituent parts. Circuits are treated as "black-boxes" which perform a function according to their specifications and communicate with adjacent circuits through input and output lines. Internal lines are hidden.

1.4 Safety and Liveness

Most properties of hardware and software systems can be classified as either a *safety* or *liveness* property. Safety properties are those where nothing bad ever happens. These include partial correctness (if the systems terminates, it has not reached any bad states), deadlock freedom, mutual exclusion (such as in an arbiter), and latch stability (e.g. if a memory device has a certain value, it will retain that value until some write action removes it). Safety properties are usually stated something like "up until some point in time, property P is true" or "from now until forever, property Q is true." Liveness properties are those where something good eventually happens. These include termination, fairness (a process will get dispatched eventually), and responsiveness (e.g. if the bus is requested, it will eventually be granted). Liveness is usually expressed "eventually property R will be true."

1.5 Theorem Prover

Propositional Temporal Logic (PTL) is the primary assertion language. This is propositional calculus (PC) with the 4 temporal operators added. It is a decidable language, and tableaux procedures are provided in [Wol84]. Frank Yellin at Stanford wrote a theorem prover which implements the decision procedure for PTL. We will refer to it as the Decision Procedure or just DP. This became available roughly midway into this research and eliminated much of the need for hand proofs of temporal logic theorems.

The notions of *validity* and *satisfiability* used by the DP come from mathematical logic. A valid formula is one which is true in all state sequences. A satisfiable formula is one which true in at least one state sequence. The program accepts temporal formulas expressed in terms of the 4 basic operators and outputs whether the formula is valid. It does so by negating the formula and using a satisfiability algorithm [Wol84] to determine whether the negation is satisfiable. The negation is satisfiable if and only if the original formula is valid. Since PTL is complete [Gab80], any valid formula is derivable in the PTL deductive system and is therefore a theorem.

1.6 Contributions of this Thesis

We introduce PTL timing diagrams to address the need to informally reason about temporal logic formulas, especially those involving the "until" operator. They are a conceptual tool rather than a formal tool. We demonstrate their use in deciding whether a temporal logic formula looks reasonable.

The proof rules for steady-state properties are the product of a syntax-directed proof method introduced here. The proof rules for global-time properties come out of an extension of the steady-state method, the global-time method. We prove the proof rules and demonstrate their application to formal reasoning about circuits.

The theory of the Execution Graph Method involves a collection algorithms and theorems. We present the theory and demonstrate the application of it in a correctness proof of a self-timed asynchronous pipeline.

In addition, several other small circuit proofs and original temporal logic theorems are presented in detail.

1.7 Organization of this Thesis

Chapter 2 describes temporal logic, some of its theorems, and its use in hardware verification. Chapter 3 discusses previous related work in hardware verification. Chapter 4 is an introduction to the PTL proof process. Steady-state properties are considered and proof rules are introduced and demonstrated. Chapter 5 extends the steady-state method to global-time properties. Forward and backward reasoning are introduced, as well as PTL timing diagrams. Chapter 6 discusses the execution graph method for proving safety and liveness properties. Chapter 7 considers anomalous behavior. Finally, Chapter 8 discusses conclusions and remaining problems.

CHAPTER 2

Temporal Logic

We introduce temporal logic. First we give a definition in terms of its syntax and semantics, then an axiomatization in line with the current temporal logic literature [Man81, Gab80, Wol83], followed by useful operator properties in the form of theorems. Then we provide examples of its use in hardware specification.

2.1 Definition of Temporal Logic

Temporal logic is classical first-order logic augmented with four temporal operators: $\Box$ (henceforth), ∇ (eventually), $\bigcirc$ (next), and U (until). The formula $\Box w$ intuitively means that w is true now and at all times in the future. ∇w means that there is a time in the future when w will hold. $\bigcirc w$ means that at the next point is time w is true. $w_1 \, U \, w_2$ means that w_1 is true now and up to, but not necessarily including, the point in time when w_2 holds; if w_2 never becomes true, w_1 is true forever. Later we add an additional operator, U_A, for convenience. $w_1 \, U_A \, w_2$ abbreviates $w_1 \, U \, (w_1 \,\&\, w_2)$. $w_1 \, U_A \, w_2$ means that w_1 is true now and up to *and including* the point in time when w_2 holds. Again if w_2 never becomes true, w_1 is true forever.

Two types of underlying models for temporal logic are *linear-time* and *branching-time*. Both represent time as going from now infinitely far into the future. Linear-time is a deterministic viewpoint. Every state has only one possible next state which can be determined. Time appears to be an infinite sequence of states. Branching-time is a nondeterministic viewpoint. Every state has a set of possible next states. Time appears to be an infinitely long branching tree beginning at the root, corresponding to "now." In this thesis we deal only with linear-time temporal logic.

We deal primarily with *propositional temporal logic (PTL)*, an extension of propositional logic or propositional calculus (PC) which includes the temporal operators. A definition of PTL follows.

Syntax

A PTL *formula* w (in , the set of formulas) is built from the following components:

(i) atomic propositions $p_1, p_2, \ldots$

(ii) temporal operators: $\square$, ∇, $\bigcirc$, and U .

(iii) logical operators $\&$ and $\neg$.

The formulas are formed from the rules:

(i) Any atomic proposition is a formula.

(ii) If w_1 and w_2 are formulas then the following are formulas:

$$w_1 \,\&\, w_2, \neg w_1,$$

$$\square w_1,\ \nabla w_1,\ \bigcirc w_1,\ w_1 \,U\, w_2.$$

As in propositional logic the symbols $\vee$ (inclusive OR) and $\supset$ are used for abbreviation. The unary operators ($\neg$, $\Box$, ∇, and $\bigcirc$) have highest precedence. Among the binary operators, the precedence from highest to lowest is $\&$, U, $\vee$, then $\supset$. U is right-associative so that $w_1 \, U \, w_2 \, U \cdots U \, w_n$ is equivalent to $w_1 \, U \, (w_2 \, U \, (\cdots U \, w_n) \cdots)$. Parentheses are used to resolve any ambiguity.

Semantics

A *model* for a temporal logic formula is a pair (μ,Σ) where

(i) μ is a function which determines the truth value of a formula $w \in$ in a state sequence $\bar{\sigma} \in \Sigma$. μ has functionality

$$\mu: \rightarrow \Sigma \rightarrow \{\textbf{true},\textbf{false}\}$$

(ii) Σ is the set of infinite state sequences $\bar{\sigma} = <\sigma_0,\sigma_1, \cdots >$ where σ_k is the state at time k. A state is an assignment of values to atomic propositions. The notation $\bar{\sigma}^{(i)}$ is used to mean the state sequence $\bar{\sigma}$ with its first i states removed.

μ is defined over the set of formulas as follows:

(i) $\mu[\![p]\!](\bar{\sigma}) = \sigma_0[\![p]\!]$ *where* $\bar{\sigma} = <\sigma_0,\sigma_1, \cdots >$

(ii) $\mu[\![\Box w]\!](\bar{\sigma}) \equiv \forall i \geq 0 . \mu[\![w]\!](\bar{\sigma}^{(i)})$

(iii) $\mu[\![\nabla w]\!](\bar{\sigma}) \equiv \exists i \geq 0 . \mu[\![w]\!](\bar{\sigma}^{(i)})$

(iv) $\mu[\![\bigcirc w]\!](\bar{\sigma}) \equiv \mu[\![w]\!](\bar{\sigma}^{(1)})$

(v) $\mu[\![w_1 \, U \, w_2]\!](\bar{\sigma}) \equiv \forall i \geq 0 . \mu[\![w_1]\!](\bar{\sigma}^{(i)})$

$$\vee \; \exists i \geq 0 . \mu[\![w_2]\!](\bar{\sigma}^{(i)}) \; \& \; \forall j, 0 \leq j < i . \mu[\![w_1]\!](\bar{\sigma}^{(j)}) .$$

The function μ is a semantics of PTL formulas. A formula w is said to be *valid* if it is true in all models (μ,Σ) written $\models w$. We also introduce the until-after U_A

operator used as an alternative to the until operator U used in much of the temporal logic literature [Wol83, Man81, Hai82]. Until-after is more convenient when expressing safety axioms of certain hardware devices because it guarantees a point in time exists when both w_1 and w_2 are true. It is defined in terms of U.

$$w_1 \, U_A \, w_2 \equiv_{def} w_1 \, U \, (w_1 \,\&\, w_2)$$

We can also give a semantics of U_A even though it is not one of the primitive operators. The semantics can be derived from the semantics of U and the definition of U_A.

$$\mu[\![w_1 \, U_A \, w_2]\!](\bar{\sigma}) \equiv \forall i \geq 0 . \mu[\![w_1]\!](\bar{\sigma}^{(i)}) \\ \vee \; \exists i \geq 0 . \mu[\![w_2]\!](\bar{\sigma}^{(i)}) \;\&\; \forall j, 0 \leq j \leq i . \mu[\![w_1]\!](\bar{\sigma}^{(j)}) .$$

Note that the until operator is defined with j ranging over $0 \leq j < i$ rather than $0 \leq j \leq i$. Both of these until operators are weak in the sense that they do not imply the eventual occurrence of the second operand, that is, neither $w_1 \, U_A \, w_2$ nor $w_1 \, U \, w_2$ imply ∇w_2. In most of the previous temporal logic work [Man81, Hai82] the formula $w_1 \, U \, w_2$ implies ∇w_2. This strong version of until will be written U^E here to emphasize the "eventually" aspect. U^E will be defined

$$w_1 \, U^E \, w_2 \equiv_{def} (w_1 \, U \, w_2) \,\&\, \nabla w_2 .$$

In addition to the basic temporal operators, some operators will be defined for convenience. The $\leftarrow$ (assign) operator is used to state that a variable is assigned a value. In the formula $B \leftarrow A$ the value of A is copied to B over a time interval. The assign operator is defined

$$x \leftarrow w \equiv \exists d_0 . [w = d_0 \,\&\, \bigcirc (x = d_0)] .$$

(For a minute we go outside PTL and use a quantifier. The $\leftarrow$ operator is only used in the informal section on hardware description, Section 2.5 below.) The $\Rightarrow$ (entails) operator, when used in a formula such as $w_1 \Rightarrow w_2$, abbreviates the *entailment* formula $\Box(w_1 \supset w_2)$. Entailment is a concept introduced in [Hug68] meaning that w_2 can be inferred from w_1 throughout time. The $\twoheadrightarrow$ (yields) operator states a liveness relationship between two formulas. $w_1 \twoheadrightarrow w_2$ is an abbreviation for $\Box(w_1 \supset \nabla w_2)$, a relationship between two formulas known as an *eventuality*. It says that, throughout time, if w_1 becomes true then w_2 must eventually become true. The $\uparrow$ (high-transition) operator states that a formula changes from false to true over one time-interval. The $\uparrow$ operator in $\uparrow w$ is an abbreviation for $\neg w \; \& \; \bigcirc w$. Similarly the $\downarrow$ (low-transition) operator used in $\downarrow w$ is an abbreviation for $w \; \& \; \bigcirc \neg w$. We will also use the expression $\sum_i w_i = 1$ to say exactly one of the w_i is true.

2.2 Axiomatization of Temporal Logic

Gabbay, Pnueli, Shelah, and Stavi [Gab80] give a complete axiomatization of PTL with the strong until operator U^E. Wolper [Wol83] gives a different axiomatization based on the weak until operator U.

Axioms

$$\vdash \nabla w \equiv \neg \Box \neg w \tag{A1}$$

$$\vdash \Box(w_1 \supset w_2) \supset (\Box w_1 \supset \Box w_2) \tag{A2}$$

$$\vdash \bigcirc \neg w \equiv \neg \bigcirc w \tag{A3}$$

$$\vdash \bigcirc(w_1 \supset w_2) \supset (\bigcirc w_1 \supset \bigcirc w_2) \tag{A4}$$

$$\vdash \Box w \supset w \;\&\; \bigcirc w \;\&\; \bigcirc \Box w \qquad \text{(A5)}$$

$$\vdash \Box (w \supset \bigcirc w) \supset (w \supset \Box w) \qquad \text{(A6)}$$

$$\vdash \Box w_1 \supset w_1 \, U \, w_2 \qquad \text{(A7)}$$

$$\vdash w_1 \, U \, w_2 \equiv w_2 \vee (w_1 \;\&\; \bigcirc(w_1 \, U \, w_2)) \qquad \text{(A8)}$$

Inference rules

If w *is a propositional tautology, then* $\vdash w$. (R1)

If $\vdash w_1 \supset w_2$ *and* $\vdash w_1$, *then* $\vdash w_2$ (R2)

If $\vdash w$, *then* $\vdash \Box w$ (R3)

The $\vdash$ symbol preceding a formula as in $\vdash w$ means that w is an axiom or theorem which is derivable within the system. $\vdash w$ may be read "w is derivable" or "w is provable." Axiom A1 states that if it is not the case that at all times w is false, then w must eventually become true. Axiom A2 says that if w_1 entails w_2 then $\Box w_2$ can be inferred from $\Box w_1$. Axiom A3 defines how the $\bigcirc$ and $\neg$ operators may be transposed. The one-way distributivity of $\bigcirc$ over $\supset$ is stated in axiom A4. Axiom A5 gives three inferences possible from $\Box w$. Individually these are:

$$\vdash \Box w \supset w$$

$$\vdash \Box w \supset \bigcirc w$$

$$\vdash \Box w \supset \bigcirc \Box w$$

Axiom A6 is the "computational induction" axiom, stating that if a property is inherited over all states, then the property being true in the inital state suffices to deduce that the property is invariantly true. Axioms A7 and A8 define weak until. Axiom A7 states that U is weak since $\Box w_1$ is sufficient to infer that $w_1 \, U \, w_2$, and that w_2 need never occur. Axiom A8 is an infinite recursive definition of the operator.

Wolper's axiomatization replaces U axiom A8 with

$$\vdash u \;\&\; \Box(u \supset (p \;\&\; \bigcirc u \vee q)) \supset p \; U \; q \tag{A9}$$

$$\vdash p \; U \; q \supset (q \vee p \;\&\; \bigcirc(p \; U \; q)) \tag{A10}$$

A9 is quite useful for proving $p \; U \; q$. The axiom is a form of induction with u as the basis and $u \supset (p \;\&\; \bigcirc u \vee q)$ as the induction hypothesis.

Rule R1 includes all tautologies of propositional logic as theorems. Temporal versions of propositional tautologies are also theorems. For example, $\Box w \supset \Box w$, the temporal version of the tautology $p \supset p$, is a theorem. Rule R2 is the rule of modus ponens. Rule R3 holds since if $\vdash w$ then w is true in all models (μ, Σ) (and all state sequences $\bar{\sigma}$), hence it is true throughout time. Rules R1 through R3 are sound.

The notion of propositional reasoning is used to obtain inference rules from theorems. Propositional reasoning is discussed in [Man81]. The idea is that if

$$\vdash (w_1 \;\&\; w_2 \;\&\; \cdots \;\&\; w_n) \supset w$$

and if $\vdash w_1$ through $\vdash w_n$ then $\vdash w$. This is stated

$$\frac{\vdash (w_1 \;\&\; w_2 \;\&\; \cdots \;\&\; w_n) \supset w \;,\; \vdash w_1 \;,\; \vdash w_2 \;,\; \cdots \;,\; \vdash w_n}{\vdash w} \tag{PR}$$

Rule PR is, in a sense, a meta-inference rule which makes inference rules out of

theorems. When rule PR is used, the antecedent

$$\vdash (w_1 \,\&\, w_2 \,\&\, \cdots \,\&\, w_n) \supset w$$

is usually omitted and is implicitly assumed. The justification for rule PR is the propositional tautology

$$\vdash [(w_1 \,\&\, w_2 \,\&\, \cdots \,\&\, w_n) \supset w] \supset [w_1 \supset (w_2 \supset (\cdots (w_n \supset w) \cdots))].$$

Applying modus ponens (rule R2) to this $n+1$ times yields $\vdash w$.

Manna's temporal logic axiomatization for sequential program verification includes a few other derived rules which we will borrow [Man81]. The equivalence rule (ER), which allows substitution of equivalent formulas, is proved by induction on the structure of formulas and is stated as follows.

Let w' be the result of replacing an occurrence of a subformula v_1 in w by v_2. Then

$$\frac{\vdash v_1 \equiv v_2}{\vdash w \equiv w'} \qquad \text{(ER)}$$

The $\Box$ operator can be introduced or eliminated in formulas. Henceforth-introduction ($\Box$I) is done by rule R3.

$$\frac{\vdash w}{\vdash \Box w} \qquad (\Box\text{I})$$

Henceforth-elimination ($\Box$E) is done by using the following rule, justified by the axiom A5, stating $\vdash \Box w \supset w$, and propositional reasoning.

$$\frac{\vdash \Box w}{\vdash w} \qquad (\Box\text{E})$$

Derivable implications can be tranformed with the $\Box\ \Box$ and $\nabla\ \nabla$ Rules:

$$\frac{\vdash w_1 \supset w_2}{\vdash \Box w_1 \supset \Box w_2}. \tag{$\Box\Box$}$$

$$\frac{\vdash w_1 \supset w_2}{\vdash \nabla w_1 \supset \nabla w_2}. \tag{$\nabla\nabla$}$$

Throughout we will be deriving assertions s which are specifications for a given circuit. s will usually be proved from some initial assumptions which we (somewhat confusingly) call axioms. These differ from the temporal logic axioms. The temporal logic axioms state a transformation of formulas and can be proven sound with respect to the semantics. The gate axioms dictate the behavior of gates and other atomic devices and cannot be proven sound without inventing a semantics of gates, which would simply be tailored to proving the axioms sound.

Proving a specification s with respect to assumption G is written

$$G \vdash s$$

In temporal logic the deduction theorem takes one of two forms: [Man81]

$$\frac{w_1 \vdash w_2}{\vdash (\Box w_1) \supset w_2}, \tag{DED}$$

the temporal logic version of the predicate logic deduction theorem, or

$$\frac{w_1 \vdash w_2}{\vdash w_1 \supset w_2}, \tag{RDED}$$

where no application of $\Box I$ *was used in any step in proving* $w_1 \vdash w_2$,

a restricted version. We use DED primarily because of lack of restriction on the deduction of $\vdash w_2$. $G \vdash s$ can be converted to the implication $\vdash G \supset s$ by the following process.

Formulate all gate axioms as holding "forever." This means inserting a $\Box$ in front of all axioms. If the conjunction of gate axioms is G then

$$G \equiv (\Box G_1) \& \cdots \& (\Box G_n)$$

for gate axioms $G_1, \cdots, G_n$. Then if

$$G \vdash s$$

then

$(\Box G_1) \& \cdots \& (\Box G_n) \vdash s$	by def. of G
$\vdash \Box (\Box G_1) \& \cdots \& (\Box G_n) \supset s$	by DED
$\vdash \Box \Box (G_1 \& \cdots \& G_n) \supset s$	by theorem T4 of Section 2.4 below
$\vdash \Box (G_1 \& \cdots \& G_n) \supset s$	by ER using T1 of Section 2.4 below

and, finally,

$$\vdash G \supset s$$

by the definition of G. Stating the axioms as "forever" formulas allows us to acheive $G \vdash s$ without the restrictions of RDED.

2.3 PTL Timing Diagrams

PTL timing diagrams are introduced in this thesis to reason whether a formula expressed in PTL is likely to be a theorem. A timing diagram is not a proof technique; for that we turn to the PTL tableaux decision procedure [Gab80, Wol83]. It is an informal, preliminary way of convincing oneself that the formula would be a theorem if tested on the decision procedure. Formulas involving U are the hardest to intuitively reason about, and it is primarily these for which the technique is used.

For linear-time PTL we can picture the formula $(w_1 \; U \; w_2)$ as being satisfied by the model of the timing diagram in Figure 2.1.

Figure 2.1. Model for $(w_1 \; U \; w_2)$

The nodes are instances of time or states (interchangeably). One very useful property of U is

$$\vdash (w_1 \; U \; w_2) \;\&\; (w_3 \; U \; w_4) \;\&\; \Box \neg (w_2 \;\&\; w_3) \supset (w_1 \; U \; w_4) \qquad (2.1)$$

The PTL timing diagram for this implication is in Figure 2.2. The arrow from the node labelled w_3 to the one labelled w_2 shows the dependency of the event of w_2 occurring on the event of w_3 no longer being in effect. Since $\Box \neg (w_2 \;\&\; w_3)$, w_2 and w_3 are in conflict throughout time, and w_2 cannot occur until w_3 is no longer in effect. The diagram shows the one possible situation, but it allows the case when w_2 takes much longer to occur. The rightward arrow says that w_2 could take much longer

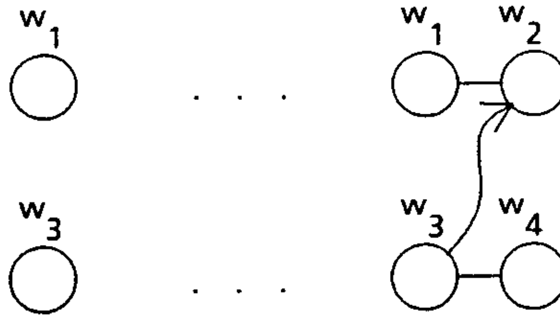


Figure 2.2 Timing diagram for Implication (2.1)

(possibly forever) to occur, but it must at least wait for w_3. So, in a sense, w_3 is *allowing* w_2 to occur.

The diagram does not show the cases where w_2 and w_4 never occur, which is allowed with our use of the weak U operator. It so happens that those situations do not invalidate the theoremhood of the formula, but the timing diagram gives no indication of this fact. We intend not to address these "forever" cases in the diagrams, since complexity of the diagrams would be too great for their usefulness. In practice, we have found that if a formula has all weak U or all strong U^E operators, which is usually the case, then the extra "forever" cases are insignificant.

The diagrams consist of nodes, which represent points in time or states where formulas hold, and arrows which indicate the partial ordering of the nodes. Many more examples occur in the subsequent sections.

2.4 PTL Theorems

Now we present some useful PTL theorems. The reader may want to skip some of them (or, at least, their proofs) until they are referenced later in the thesis. If the proof was done in the PTL deductive system, then a "$\vdash$" precedes the theorem. If the proof was done in the PTL semantics either by hand or by the decision procedure, then

a "$\models$" precedes the theorem. By the completeness of the PTL deductive system [Gab80], any theorem proved valid in the semantics is also derivable in the deductive system. Many of the theorems have been proved elsewhere in which case the reference is given.

$$\vdash \Box w_1 \vee \Box w_2 \supset \Box(w_1 \vee w_2) \qquad \text{(T0)}$$

Theorem 25 of [Man81a].

$$\vdash \Box w \equiv \Box \Box w \qquad \text{(T1)}$$

T2 of [Man81].

$$\vdash \nabla w \equiv \nabla \nabla w \qquad \text{(T2)}$$

T3 of [Man81].

$$\vdash \Box(w_1 \supset w_2) \supset (\nabla w_1 \supset \nabla w_2) \qquad \text{(T3)}$$

T5 of [Man81].

$$\vdash \Box(w_1 \,\&\, w_2) \equiv (\Box w_1 \,\&\, \Box w_2) \qquad \text{(T4)}$$

T6 of [Man81].

$$\vdash \nabla (w_1 \vee w_2) \equiv (\nabla w_1 \vee \nabla w_2) \tag{T5}$$

T7 of [Man81].

$$\vdash \nabla (w_1 \,\&\, w_2) \supset \nabla w_1 \,\&\, \nabla w_2 \tag{T6}$$

T8 of [Man81].

$$\vdash (\Box w_1 \,\&\, \nabla w_2) \supset \nabla (w_1 \,\&\, w_2) \tag{T7}$$

T10 of [Man81].

$$\vdash \nabla \Box w_1 \,\&\, \nabla \Box w_2 \equiv \nabla \Box (w_1 \,\&\, w_2) \tag{T8}$$

Theorem 18 of [Hai82].

$$\models w_1 \, U^E \, w_2 \equiv (w_1 \, U \, w_2) \,\&\, \nabla w_2 \tag{T9}$$

This is by the semantics of U^E which is

$$\mu[\![w_1 \, U^E \, w_2]\!](\bar{\sigma}) \equiv \exists i \geq 0 . \mu[\![w_2]\!](\bar{\sigma}^{(i)}) \;\&\; \forall j, 0 \leq j < i . \mu[\![w_1]\!](\bar{\sigma}^{(j)})$$

and the semantics of U .

$$\models w_1 \, U \, w_2 \equiv \Box w_1 \vee (w_1 \, U^E \, w_2) \tag{T10}$$

This is by the semantics of U^E and U .

$$\vdash w_1 \, U \, w_2 \equiv \Box w_1 \vee ((w_1 \, U \, w_2) \,\&\, \nabla w_1) \tag{T11}$$

This is by T9 and T10.

$$\vdash w_1 \, U_A \, w_2 \equiv \Box w_1 \vee ((w_1 \, U_A \, w_2) \,\&\, \nabla w_1) \tag{T12}$$

This is by T9 and T10.

$$\models (w_1 \, U \, w_2) \,\&\, \Box (w_1 \supset w_3) \supset (w_2 \, U \, w_3) \tag{T13}$$

Proved on the DP.

$$\models (w_1 \, U \, w_2) \,\&\, \Box (w_2 \supset w_3) \supset (w_1 \, U \, w_3) \tag{T14}$$

Proved on the DP.

$$\begin{aligned} &\models (w_1 \, U \ \cdots \ U \, w_n) \,\&\, \\ &\quad \Box (w_1 \supset w_1') \,\&\, \cdots \,\&\, \Box (w_n \supset w_n') \supset \\ &\quad (w_1' \, U \ \cdots \ U \, w_n') \end{aligned} \tag{T15}$$

Proved on the DP.

$$\begin{aligned} &\models (w_1 \, U^E \ \cdots \ U^E \, w_n) \,\&\, \\ &\quad \Box (w_1 \supset w_1') \,\&\, \cdots \,\&\, \Box (w_n \supset w_n') \supset \end{aligned} \tag{T16}$$

$$(w_1{}' \, U^E \cdots U^E \, w_n{}')$$

Proved on the DP.

$$\models \bigcirc \Box w \equiv \Box \bigcirc w \tag{T17}$$

T17 of [Man81].

$$\models \bigcirc \nabla w \equiv \nabla \bigcirc w \tag{T18}$$

T18 of [Man81].

$$\models \Box w \equiv (w \ \& \ \bigcirc \Box w) \tag{T19}$$

T19 of [Man81].

$$\models \nabla w \equiv (w \vee \bigcirc \nabla w) \tag{T20}$$

T20 of [Man81].

$$\models \bigcirc(w_1 \ \& \ w_2) \equiv (\bigcirc w_1 \ \& \ \bigcirc w_2) \tag{T21}$$

Theorem 19 of [Man81a].

$$\models \bigcirc(w_1 \vee w_2) \equiv (\bigcirc w_1 \vee \bigcirc w_2) \tag{T22}$$

Theorem 20 of [Man81a].

$$\models \bigcirc(w_1 \supset w_2) \equiv (\bigcirc w_1 \supset \bigcirc w_2) \tag{T23}$$

Theorem 21 of [Man81a].

$$\models \bigcirc(w_1 \equiv w_2) \equiv (\bigcirc w_1 \equiv \bigcirc w_2) \tag{T24}$$

Theorem 22 of [Man81a].

$$\models \Box(w_1 \supset w_2) \supset (\neg w_1 \, U \, w_2) \tag{T25}$$

Proved on the DP.

$$\models w_1 \, U \, (w_1 \, U \, w_2) \supset w_1 \, U \, w_2 \tag{T26}$$

Proved on the DP.

$$\models w_1 \, U \, w_1 \, U \, \cdots \, U \, w_1 \, U \, w_2 \supset w_1 \, U \, w_2 \tag{T27}$$

Proved on the DP.

$$\models (w_1 \,\&\, w_2) \, U_\Lambda \, w_3 \equiv (w_1 \, U_\Lambda \, w_2) \,\&\, (w_1 \, U_\Lambda \, w_3) \tag{T28}$$

Proved on the DP.

$$\models w_1\, U_A\, (w_2 \vee w_3) \equiv (w_1\, U_A\, w_2) \vee (w_1\, U_A\, w_3) \qquad \text{(T29)}$$

Proved on the DP.

$$\models (w_1\, U\, w_2)\ \&\ (w_3\, U\, w_4)\ \&\ \Box \neg (w_2\ \&\ w_3) \supset (w_1\, U\, w_4) \qquad \text{(T30)}$$

from implication (2.1) above. Proved on the DP.

$$\vdash w \supset \nabla w \qquad \text{(T31)}$$

T1 of [Man81].

2.5 Describing Hardware with Temporal Logic

Temporal logic is a natural language for describing static and dynamic properties of hardware systems. The ability to talk about time in both the specific sense (e.g. "at time i") and abstract sense (e.g. "at some point in the future") make it a powerful reasoning language. In a sense, temporal logic is a formalization of the timing diagrams usually associated with describing circuit behavior. It may be used as both a hardware description language to describe implementations and an assertion language to describe properties and behavior.

2.5.1 Signal Specification

The shape of a signal may be specified. Consider, for example, a device with a $\overline{Reset}$ line which is active low. The device may be a flip-flop, a register, or a computational unit. $\overline{Reset}$ is pulled low in order to reset the device. The formula

$$\downarrow \overline{Reset} \;\&\; \bigcirc\uparrow \overline{Reset} \;\&\; \bigcirc^2 \,\Box \overline{Reset}. \tag{2.3}$$

specifies that $\overline{Reset}$ makes a low-transition followed by a high-transition, as shown in Figure 2.3. This formula, expanded based on the definitions of the $\uparrow$ and $\downarrow$ operators, is equivalent to

$$(\overline{Reset} \;\&\; \bigcirc\neg \overline{Reset}) \;\&\; \bigcirc(\neg \overline{Reset} \;\&\; \bigcirc\overline{Reset}) \;\&\; \bigcirc^2 \,\Box \overline{Reset}$$

which may be simplified to

$$\overline{Reset} \;\&\; \bigcirc\neg \overline{Reset} \;\&\; \bigcirc^2 \,\Box \overline{Reset}. \tag{2.4}$$

Both Formula 2.3 and Formula 2.4 describe the signal in Figure 2.3. Either may be used as an assertion of how the reset signal is to behave for the device to be properly reset.

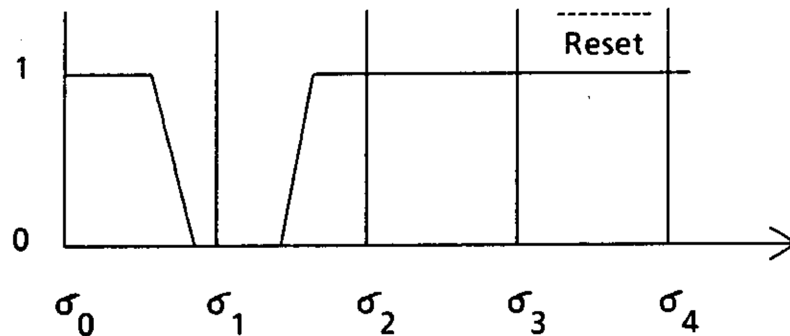


Figure 2.3 A Reset signal to a device

2.5.2 Memory Properties

Safety and liveness properties of memory devices may be specified. The operation of the register $R0$ in Figure 2.4 may be specified in temporal logic by the pair of formulas

$$R0 = x \implies R0 = x \; \mathcal{U} \; LoadR0 \qquad \text{(R0safe)}$$

$$\uparrow LoadR0 \implies [\, R0 \leftarrow Selector[0] \,] \qquad \text{(R0live)}$$

where $Selector[0]$ is the selector's output line 0.

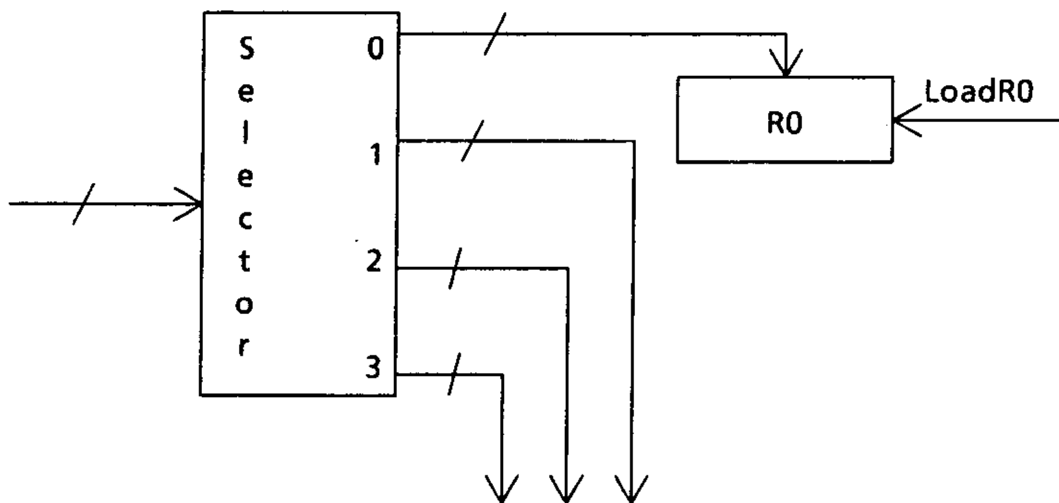


Figure 2.4 A Register and Associated Circuitry

Formula R0safe is a safety assertion, stating that $R0$ will retain its value until the $LoadR0$ line goes high. Formula R0live is a liveness assertion, stating that when the $LoadR0$ line goes high, $R0$ will accept a new value, the selector output value. The timing of the load operation is shown in Figure 2.5.

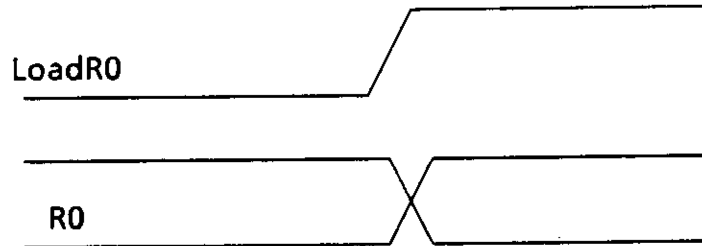


Figure 2.5 Timing of the Register Load Operation

2.3.3 Asynchronous Communication

Asynchronous communication can be specified using safety and liveness assertions. Consider a bus arbiter with a request line *Req* and acknowledgement line *Ack* shown in Figure 2.6. Two safety assertions ensure that the *Req* line is held in the proper state until an acknowledgement is received.

$$Req \;\Rightarrow\; Req \;U_A\; Ack \qquad \text{(ArbSafe1)}$$

$$\neg Req \;\Rightarrow\; \neg Req \;U_A\; \neg Ack \qquad \text{(ArbSafe2)}$$

Two more safety assertions ensure that the *Ack* line responds properly to the *Req* line.

$$Ack \;\Rightarrow\; Ack \;U_A\; \neg Req \qquad \text{(ArbSafe3)}$$

$$\neg Ack \;\Rightarrow\; \neg Ack \;U_A\; Req \qquad \text{(ArbSafe4)}$$

Two liveness assertions ensure that the requests are eventually followed by

acknowledgements. This guarantees responsiveness of the arbiter.

$$Req \twoheadrightarrow Ack \qquad \text{(ArbLive1)}$$

$$\neg Req \twoheadrightarrow \neg Ack \qquad \text{(ArbLive2)}$$

The last liveness assertion ensures that the user cannot hold the arbiter indefinitely without releasing it with $\downarrow Req$.

$$Req \twoheadrightarrow \neg Req \qquad \text{(ArbLive3)}$$

The timing of the bus arbiter Req and Ack lines is shown in Figure 2.7.

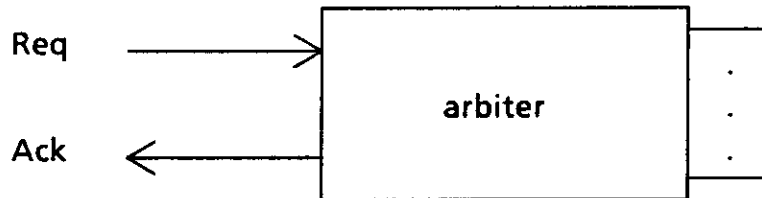


Figure 2.6 A Bus Arbiter and Control Lines

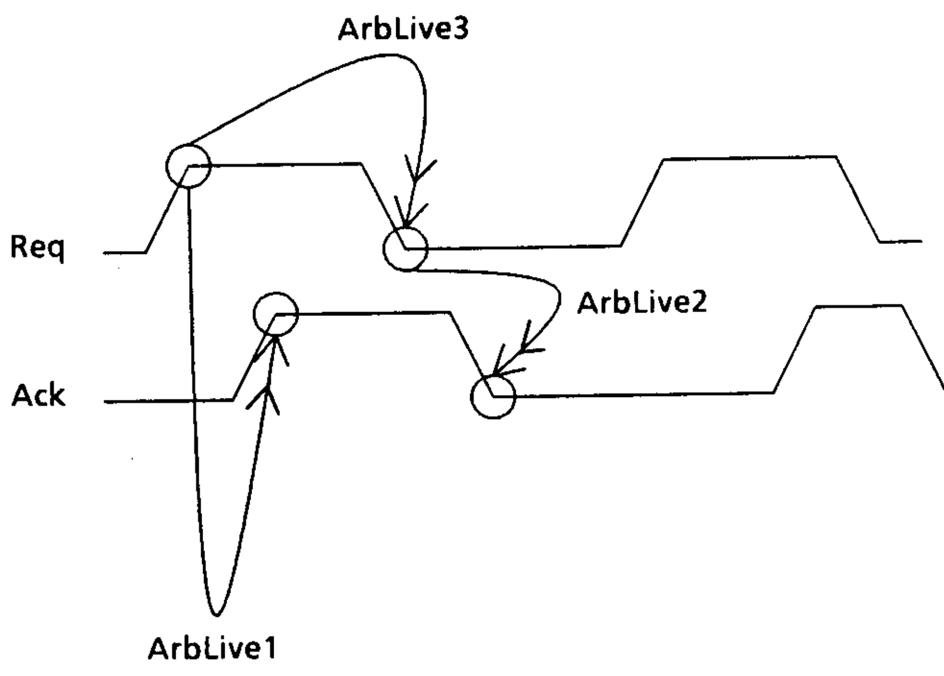


Figure 2.7 Timing of Bus Arbiter Control Lines

CHAPTER 3

Related Work

Recently hardware verification has been getting more attention in the literature. This has been accelerated by the increasing number of hardware description languages and design automation systems. The work varies in nature from the theoretical foundations to fully implemented verifiers.

Research in hardware verification may be classified into two approaches: *semantics-based* verification, where one is concerned with what a circuit *is*, and *logic-based* verification, where one is primarily concerned with what *properties* a circuit or a portion thereof *possesses*. The logic-based work can be further classified as predicate logic-based and temporal logic-based, the latter being closest to this work.

3.1 Semantics-Based Verification

Semantics-based verification is a model-theoretic approach. It hinges on finding a suitable mathematical model which can explain all aspects of interest. Formal reasoning is done within the model and it is assumed that all objects and operations in the model correctly represent circuit behavior. As is the case with proofs in denotational

semantics, semantics-based proofs can require a rather complicated manipulation since all information about the behavior of a device must be available in the mathematical object denoted by the circuit. Gordon [Gor81a, Gor81b, Gor81c] and Milne [Mil82, Mil83a, Mil83b] are the primary proponents of this approach, which was inspired by Milner's semantics of concurrent processes [Mil80].

3.1.1 Milner's CCS

Calculus of Communicating Systems (CCS) [Mil80] is a semantics of communicating processes which has stimulated further work in the semantics of hardware. With CCS each process in the system to be modelled is roughly a finite state machine which can connect and communicate with the other processes. A CCS system is usually described by equations, each equation describing a process. Since CCS is a semantic meta-language much like denotational semantics [Sto77], the syntactic contructs of a programming language can be translated into CCS expressions, allowing one to formally describe and reason about their behavior.

The primary primitive concepts which give CCS its modelling power are value-transmission or message-passing, the notion of a port, and the rules governing how processes connect and interact. Each process has a set of input ports and output ports. Output port names are distinguished from input port names by having a bar over them (e.q. $\bar{\alpha}$). An output port of one process may connect to an input port of another when the two processes are composed if the two ports have the same name (except for the bar). If two ports are connected by composition then the output value of one process is the input value of the other, as in the message-passing found in Hoare's Communicating Sequential Processes [Hoa78].

A CCS expression denotes a CCS behavior. The binding of variables in expressions is similar to λ-calculus. Each port name α which precedes a variable as in $\alpha x.E(x)$ may be thought of as the λ in λ-expressions. Here x becomes a bound variable in the expression $E(x)$. In CCS, however, the bound variable value may be either an input value, just as in λ-calculus, or an output value with the side-effect of being sent out through the output port.

The primary operators are composition, alternation, and restriction. If $\mathbf{B}_L$ is the domain of CCS behaviors with the set of port names L, then the operators have functionality

composition: $\quad | : \mathbf{B}_L \times \mathbf{B}_M \rightarrow \mathbf{B}_{L \cup M}$

alternation: $\quad + : \mathbf{B}_L \times \mathbf{B}_M \rightarrow \mathbf{B}_{L \cup M}$

restriction: $\quad /\alpha: \mathbf{B}_L \rightarrow \mathbf{B}_{L-\{\alpha,\bar{\alpha}\}}$

Composition is the port connection and parallel composite behavior of two processes. It can be applied repeatedly, building large networks of processes. Processes which are composed operate in parallel and communicate though the ports. Alternation is the nondeterministic selection of two or more behaviors. Each of the behaviors usually has at least one guard with the port name and an optional bound variable (as in $\alpha x.E(x)$). Restriction is used to hide port names from the composition operation. If a port is hidden by restriction it cannot connect with other ports with the same name in other processes. The behavior of a real-time system demonstrates the use of the three operators.

Figure 3.1 illustrates a real-time system of three processes communicating through the ports $\alpha,\bar{\alpha},\beta$, and $\bar{\beta}$. The behavior of the SENDER process is expressed

recursively:

$$SENDER(x) = \alpha.\overline{\beta}x.SENDER(x) + \gamma y.SENDER(y).$$

SENDER stores a value and sends it (through $\overline{\beta}$) to RECEIVER periodically, i.e., when signalled by ALARM (through α). If SENDER receives a new value y through input port γ, this becomes the new stored value. The '+' operator denotes alternation: either action can take place nondeterministically. The RECEIVER process accepts an alarm signal, then accepts a value from SENDER, then calls procedure P to process the value. P takes RECEIVER as an argument so that it can call it when it is done processing. The behavior is

$$RECEIVER = \alpha.\beta z.P(z, RECEIVER).$$

ALARM counts like a clock continuously. Every 100th tick it sends an alarm signal out to the other processes. ALARM is expressed

$$ALARM(i) = \textbf{if } i = 100 \textbf{ then } \overline{\alpha}.ALARM(1) \textbf{ else } ALARM(i+1).$$

The behavior of the entire system is

$$(SENDER(x_0) \mid RECEIVER \mid ALARM(0))/\alpha/\beta.$$

CCS can also be used to model hardware systems by considering each device as a process. Ports are used to represent lines and message-passing may be thought of as signal transmission. For example, a multiplexor can be described

$$MUX = \alpha_{select}.\alpha x.\overline{\gamma}x.MUX + \beta_{select}.\beta x.\overline{\gamma}x.MUX$$

meaning that if either an α_{select} or β_{select} signal is received a value is accepted through

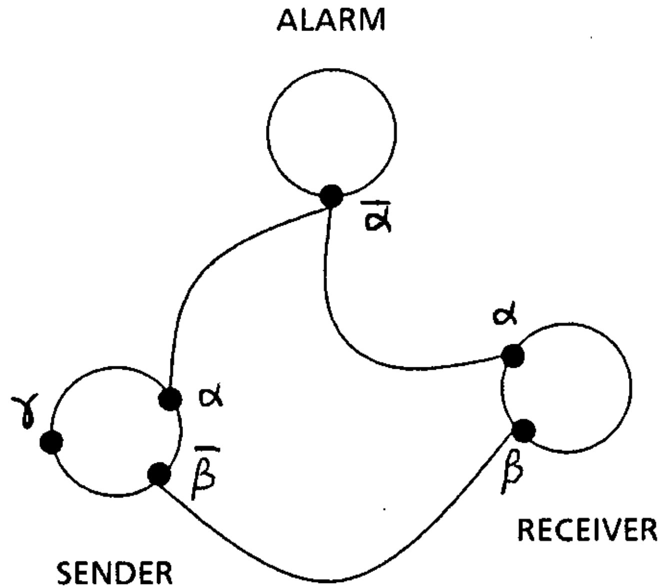


Figure 3.1 CCS Model of 3 Real-Time Processes

α or β respectively and passed on through $\bar{\gamma}$. When clocked, a register outputs the current value x and accepts and stores a new value.

$$REG(x) = \gamma_{clock}.\bar{\delta}x.\gamma y.REG(y)$$

An incrementer increments the input and passes it on.

$$INC = \delta x.\bar{\beta}(x+1).INC$$

Composing the three devices yields a system whereby a value can be loaded with α_{select} then repeatedly incremented with γ_{clock} and β_{select}. $\bar{\delta}$ is where the output is taken from. Figure 3.2 shows the CCS model of the circuit. The complete circuit description is

$$(MUX \mid REG(x_0) \mid INC)/\alpha/\beta$$

where x_0 is the inital register contents.

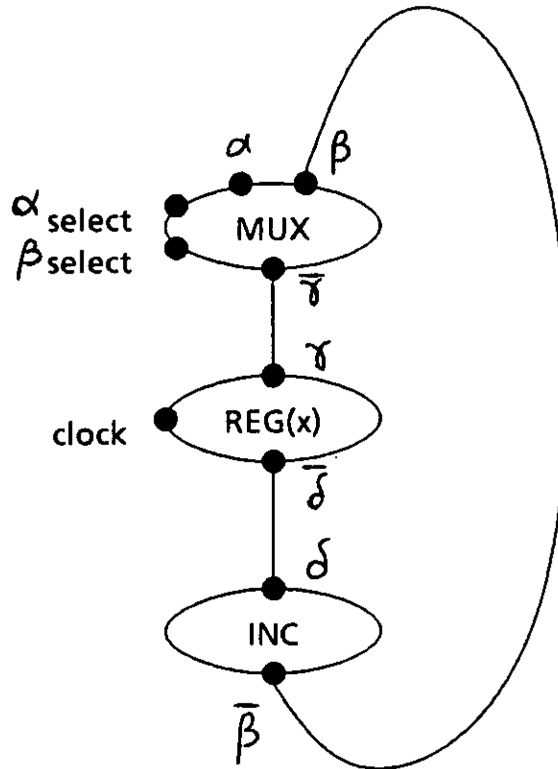


Figure 3.2 CCS Model of a Simple Register-Transfer System

Devices modelled as CCS processes are active entities which are unlike either combinational or sequential devices. The CCS multiplexor must wait for a select signal before the value can pass through. An actual combinational device is a function of its inputs which is continuously operational. The CCS register outputs the stored value once per clocking. It should output it continuously or, at least, as many times as needed. These deficiencies are addressed by Gordon in his synchonous hardware model.

3.1.2 Gordon's Synchronous Hardware Model

Gordon's model of register transfer systems [Gor81a, Gor81b, Gor81c] is a semantics of hardware which extends the CCS framework. The notions of composition, restriction, and ports are roughly as in CCS, but Gordon's model is of a more synchronous nature. The model is used to verify circuits using a synthesis approach. Individual devices are described by equational specifications which are assumed to be correct. When a circuit is built by composing devices, the equation describing the behavior of the composite circuit is found by consulting the model. If the resulting equation describes the correct behavior then the circuit is correct.

Gordon's model may be considered a denotational semantics of sequential systems with domains, equations, and a translation from the syntactic devices to the equations of the meta-language. Just as in the denotational approach, the semantics of composite objects is described in terms of the semantics of the constituents. Since circuits consist of two types of devices (combinational and sequential) there are two primary domains.

$$Com[X;Y] = Sig[X] \rightarrow Sig[Y]$$

$$Seq[X;Y] = Sig[X] \rightarrow (Sig[Y] \times Seq[X;Y])$$

A combinational device with input line (port) names in X ($X = \{x_1, \cdots, x_m\}$) and output line names in Y ($Y = \{y_1, \cdots, y_n\}$) belongs to $Com[X;Y]$, the domain of functions from signals on X, $Sig[X]$, to signals on Y, $Sig[Y]$. Sequential devices are inherently recursive. A sequential device is a function $(fstof) \in Com[X;Y]$ from an input value to output value until it is clocked. It then becomes a function $(snd(f(i))) \in Seq[X;Y]$ where $i \in Sig[X]$ is the input line value when clocking occurs. The sequential behavior of a sequential machine M is given by the function $\mathbf{B}[\![M]\!]$: $S_M \rightarrow Seq[X;Y]$

defined

$$\mathbf{B}[\![M]\!](\sigma) = \lambda i.(out_M(i,\sigma), \mathbf{B}[\![M]\!](next_M(i,\sigma)))$$

where σ is the internal state, i is the input value, out_M is the output function, and $next_M$ is the next-state function. This equation is the denotational semantics of sequential machines [Gor80].

A device specification equation determines the behavior of a device in all situations. The specification of a device behavior has the general form

$$N(s_1, \cdots, s_l) = \lambda\{x_1, \cdots x_m\}.(\{y_1{=}E_1, \cdots, y_n{=}E_n\}, N(E_{s_1}, \cdots, E_{s_l}))$$

where N is the device name, s_i's are state variables, x_j's are input line names, y_k's are output line names, and E's are expressions defining output and next state values. The λ in the specification indicates that x_j's are bound variables to be substituted for in the expressions by the input values. Upon clocking, the device accepts the inputs, producing the current output and next state.

Composition and restriction operations are as in CCS. Composition of two behaviors f_1 and f_2 is written $[\![f_1 \mid f_2]\!]$. The composition theorem determines the behavior of a circuit constructed using composition. When two devices are composed the output of one device becomes the input of another because of the connection of ports with the same name. The composition theorem is stated

$$[\![\lambda\{x \mid x \in X_1\}.(\{y{=}E_y \mid y \in Y_1\}, E_1) \mid \cdots \mid \lambda\{x \mid x \in X_n\}.(\{y{=}E_y \mid y \in Y_n\}, E_n)]\!]/L$$

$$= \lambda\{x \mid x \in \bigcup X_i - \bigcup Y_i\}.$$

$$letrec\ \{y{=}E_y \mid y \in (\bigcup X_i) \cap (\bigcup Y_i)\}$$

$$in\ (\{y{=}E_y \mid y \in \bigcup Y_i - L\}, [\![E_1 \mid \cdots \mid E_n]\!]/L).$$

It says that if n devices are composed and restricted by the set of ports L, then the resulting circuit has fewer input lines x $(x \in \bigcup X_i - \bigcup Y_i)$ due to internal feedback and has output expressions E_y such that $y \in (\bigcup X_i) \cap (\bigcup Y_i)$ are recursively replaced for x's in the composite expression since output lines are connected to input lines of the same name. The derived expression for the composite behavior may be judged correct by inspection or shown equivalent to another higher-level specification using a technique called simulation induction which is an induction on the number of times the two circuits are clocked. Two machines which satisfy the simulation induction theorem can effectively simulate each other in lock-step and are considered equivalent.

Consider again the system introduced in Figure 3.2 which consists of a multiplexor, register, and incrementer. In Gordon's notation the device behaviors are

$$MUX = \lambda\{select, \alpha, \beta\}.(\{\gamma{=}\textbf{if}\ select\ \textbf{then}\ \beta\ \textbf{else}\ \alpha\}, MUX)$$

$$REG(n) = \lambda\{\gamma\}.(\{\delta{=}n\}, REG(\gamma))$$

$$INC = \lambda\{\delta\}.(\{\beta{=}\delta{+}1\}, INC).$$

Figure 3.3 shows the system as described by Gordon's model. Using the composition theorem, the circuit behavior is

$COUNT(n)$

$$= \lambda\{select, \alpha\}.$$

$$letrec\ \{\gamma{=}\textbf{if}\ select\ \textbf{then}\ \beta\ \textbf{else}\ \alpha, \delta{=}n, \beta{=}\delta{+}1\}$$

$$in\ \beta{=}\delta{+}1, [\![MUX \mid REG(\gamma) \mid INC]\!]/\gamma/\delta$$

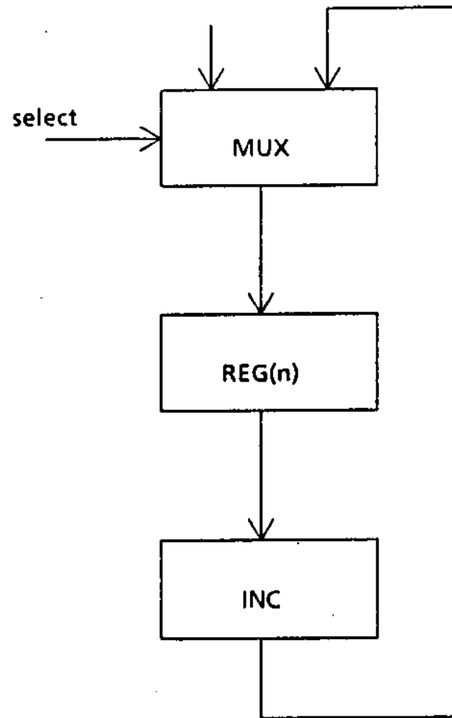


Figure 3.3 Gordon's Model of the Register-Transfer System

$= \lambda\{select, \alpha\}.(\{\beta = n+1\}, COUNT(\text{if } select \text{ then } n+1 \text{ else } \alpha)).$

which is correct (by inspection).

Nowhere in Gordon's specifications does a clock appear. A common clock is assumed to be attached to each register. This is a clean assumption, but it eliminates a class of circuits like the ripple counter, which uses the output of one stage as the clock of another.

Barrow discusses VERIFY [Bar83], a PROLOG program which implements Gordon's verification method and, in particular, the composition theorem. It performs automatic verification by composing device specifications and simplifying expressions to show that the composite behavior of the circuit is as is expected. It is claimed that VERIFY has been used to verify VLSI circuits of up to 180,000 transistors.

3.1.3 Milne's CIRCAL

Milne's CIRCAL (Circuit Calculus) [Mil82] is a CCS-based hardware model which is suitable for modelling low-level elements such as nMOS and CMOS transistors, inverters, gates, and storage elements. Ports, composition, alternation, and restriction are defined much like in CCS, but restriction is known as "abstraction" in CIRCAL, and the ports are simpler than CCS ports in that they involve only the occurrence of events and no value-transmission. Applications include silicon compiler correctness [Mil83a] and timing analysis [Mil83b].

Figure 3.4 A NOR Gate Described in CIRCAL

Since CIRCAL ports do not transmit or receive values, a two-valued line must be represented by a pair of ports: $\alpha 0, \alpha 1$. Composite ports, a CIRCAL concept not in CCS, allow a device to require that two events, α and β say, occur simultaneously. This is written, for example ($\alpha\ \beta$). The behavior of the NOR gate shown in Figure 3.4, for example, is expressed recursively as

$$N00 = (\overline{\alpha 1}\ \gamma 0)\, N10 \ + \ (\overline{\beta 1}\ \gamma 0)\, N01 \ + \ (\overline{\alpha 1}\ \overline{\beta 1}\ \gamma 0)\, N11$$

$$N10 = (\overline{\alpha 0}\ \gamma 1)\, N00 \ + \ \overline{\beta 1}\, N11 \ + \ (\overline{\alpha 0}\ \beta 1)\, N01$$

$$N01 = (\overline{\beta 0}\ \gamma 1)\, N00 \ + \ \overline{\alpha 1}\, N11 \ + \ (\overline{\alpha 1}\ \beta 0)\, N10$$

$$N11 = \overline{\alpha 0}\, N01 \; + \; \overline{\beta 0}\, N10 \; + \; (\overline{\alpha 0}\; \overline{\beta 0}\; \gamma 1)\, N00$$

where Nxy determines the behavior for α port value x and β port value y. (The bar, as in $\overline{\alpha 1}$, is used to denote input directionality.) The above equations effectively contain the truth-table for the NOR function. Since CIRCAL (unlike Gordon's model) has no logical operators or concept of a storage device, these devices must be built from the primitives. A correctness proof in CIRCAL is similar to one in CCS and Gordon's model: a proof of equivalence of two expressions.

Two primary axioms are used to derive the behavior of composite systems: the composition axiom and the abstraction axiom. The composition axiom determines the behavior of a new object consisting of the composition of two objects. The abstraction axiom determines the behavior of an object whose parts are being hidden by abstraction.

Milne uses CIRCAL to prove the functional correctness of a simple silicon compiler [Mil83a]. The compiler is a function L (for layout) which maps NOR-expressions ($n \in NE$) into a layout language (LL). Two semantic mappings, N and S, give the CIRCAL-semantics of NOR-expressions and layout language expressions respectively. The proof consists of showing that

$$\forall n \in NE.\; S(L(n)) = N(n).$$

3.2 Predicate Logic-Based Verification

Predicate logic approaches are closest to the traditional Floyd/Hoare axiomatic verification method for sequential programs. The assertion language is first-order predicate logic, axioms and inference rules are used to carry out the proofs, and timing

issues are essentially ignored The dissertations of Wagner [Wag77] and Patterson [Pat76b] are milestones in this general area.

3.2.1 Wagner's Hardware Verification System

One of the earliest efforts in hardware verification is Wagner's method which uses first-order logic as the assertion language and a non-procedural hardware description language for device and circuit specifications [Wag77]. Like the semantics-based approaches, the method is a synthesis one: individual device specifications are combined and manipulated to yield a circuit specification. The resulting specification is then checked for correctness by inspection.

The hardware description language is simple but appropriate for the application. Register transfer operations are written with the condition for transfer first which is usually a predicate about the clock or reset lines. For example, a negative edge-triggered JK flip-flop would be described

$$/ \downarrow Clk \; / \; Q \leftarrow (J \;\&\; \neg Q) \vee (\neg K \;\&\; Q); \tag{3.1}$$

$$/ \neg \downarrow Clk \; / \; Q \leftarrow Q; . \tag{3.2}$$

Line 3.1 says that, when clocked with a negative edge ($\downarrow Clk$), the flip-flop is set or reset, depending on J and K. Line 3.2 says that at all other times it retains its value.

A set of axioms are given which allow several individual assertions to be reduced to a more concise one. A software tool aids the symbolic manipulation by applying the reduction axioms, as directed by the user.

3.2.2 Patterson's STRUM

STRUM is a system developed at UCLA by Patterson for proving the correctness of microprograms [Pat76a, Pat76b]. STRUM consists of an Algol-like high-level microprogramming language, a microcode compiler, a verification condition generator, and an algebraic simplifier. The verification method is analytic. A microprogram is written with invariants and pre- and post-conditions. It is verified using the verification condition generator and algebraic simplifier. The main research contribution is experimental: demonstrating that Hoare axiomatics can be used as the basis of a useful automatic program verification system.

3.3 Temporal Logic-Based Verification

The desire to make assertions about the timing of events in circuits has lead to temporal logic as a language for writing specifications and reasoning. Conventional predicate logic must be augmented with time variables in order to deal with time. Doing this would yield a system with the expressiveness of temporal logic, but not the elegance. Temporal logic is naturally suited to dealing with time on an abstract and concrete level.

3.3.1 Malachi and Owicki's Temporal Specifications

In [Sei80] Seitz introduces self-timed VLSI circuits which operate asynchronously and generate completion signals. Malachi and Owicki [Mal81] demonstrate how temporal logic can be used to formally specify the operation of self-timed combinational logic, including pipelines and state machines.

We introduce a bit of the notation to give a feel for the specifications. Double rail codes are used to encode the data values $\{0,1,\lambda\}$, where λ is undefined. For a signal line x, let "x is defined" be abbreviated $d(x)$. For a set of lines X let

$$d(X) \equiv_{def} (\exists x \in X).d(x) \ \ and \ \ D(X) \equiv_{def} (\forall x \in X).d(x).$$

$\neg d(x)$, for example, expresses "all lines are undefined." One of the properties of a self-timed logic module is that output lines in O are undefined until at least one of the input lines in I are defined. This property can be expressed using the temporal until operator U as

$$\neg d(O) \supset \neg d(O)\, U\, d(I).$$

Another property is that sometime after all inputs I are defined, the output will be defined. This can be expressed using the temporal eventually operator ∇ as

$$D(I) \supset \nabla D(O).$$

The authors demonstrate that temporal logic is an elegant specification language. They do not, however, address formal verfication.

3.3.2 Moszkowski's ITL

Most of the temporal logic research has centered on the linear-time, four-operator version which will be called "standard" here. The concentration of effort provides a sound foundation for anyone using this particular type of temporal logic, but this type is not necessarily the best. Moszkowski introduces Interval Temporal Logic (ITL), a specification language particularly suited to the ideosyncrasies of hardware [Mos83]. Moszkowski uses ITL to specify in detail the structure and timing of adders, latches, multipliers, and ALUs. The specifications take advantage of the many ITL operators to describe aspects such as clocking, feedback, signal dependences, and propagation delay. It is intended that ITL be used as both a hardware description and assertion language.

ITL's semantics is based on finite state intervals rather than infinite state sequences. The meaning of an ITL formula w is determined by the model in a state interval of length $n : s_0, \cdots, s_n$.

$$\mu_{s_0, \cdots, s_n}[\![w]\!] \in \{\text{true}, \text{false}\}$$

This gives ITL a stricter dependence on the basic unit of time. Henceforth, for example, no longer means "now and forever" but instead "now and until the end of the time interval of interest." A *len* function returns the length of the interval of evalution

$$\mu_{s_0, \cdots, s_n}[\![len]\!] = n$$

This is generally not possible in a standard model since the state interval is infinite.

Specifications of devices are quite detailed and comprehensive, especially in the area of timing. A special predicate *struct* is provided to declare values associated with the device: the input and output line names, internal values, and performance parameters. For example, to declare predicate $SimpleDFlipFlop(F)$, which is true if F is a D flip-flop we define

$SimpleDFlipFlop(F) \equiv SimpleDFFStructure(F) \;\&\; \Box Store(F, i)$, for $i \in \{0,1\}$

where *SimpleDFFStructure* is defined

$SimpleDFFStructure(F) \equiv F : struct[$

$(Ck, D) : Bit \quad \%Inputs$

$(Q, \overline{Q}) : Bit \quad \%Outputs$

$(c1, c2, c3, hld, lat) : time \quad \%Parameters$

and *Store* is defined

$$Store(F,i) \equiv [\uparrow\downarrow^{c1,c2,c3}Ck \;\&\; Ck\ blk^{hld}\ D \;\&\; beg(D=i)]$$

$$\twoheadrightarrow [beg(Q=i \;\&\; \bar{Q}=\neg i) \;\&\; Ck\ blk^{lat} <Q,\bar{Q}>].$$

The *Store* predicate demonstrates ITL's specification of timing. The shape of the Ck pulse is specified by the $c1$, $c2$, and $c3$ parameters. $\uparrow\downarrow^{c1,c2,c3}Ck$ says that the pulse must be low for $c1$ time intervals, then high for $c2$ time intervals, then low again for $c3$ time intervals. The *blk* operator is used to say that one signal blocks another. A signal A blocks a signal B if, as long as A is stable, B is stable. The $\twoheadrightarrow$ operator is a sort of implication over time. If $A \twoheadrightarrow B$ then if A is high B must become high sometime later. Roughly speaking, the *Store* predicate says that if D has the value i when clocked, Q will have the value i after being clocked.

The main results of Moszkowski's dissertation are

(i) the demonstration of ITL, a new temporal logic which is rich enough to express detailed timing requirements of hardware deviced currently made in industry,

(ii) theoretical complexity results regarding ITL,

(iii) specifications of several conventional hardware devices.

The same features which give ITL its expressive power can make it cumbersome in the proof process. There are many more operators to deal with, and so the number of possible theorems is quite large. Automating the proving of ITL theorems is an ambitious research project which would offer a solution.

3.3.3 Bochmann's Arbiter Proof

Bochmann was the first to use temporal logic when formally verifying hardware by proof construction [Boc82]. Bochmann verifies the arbiter appearing in [Sei80b] by a reachability analysis of the interal states. Given several assumptions about the arbiter's environment and the behavior of its constituent devices, safety and liveness properties are shown to hold.

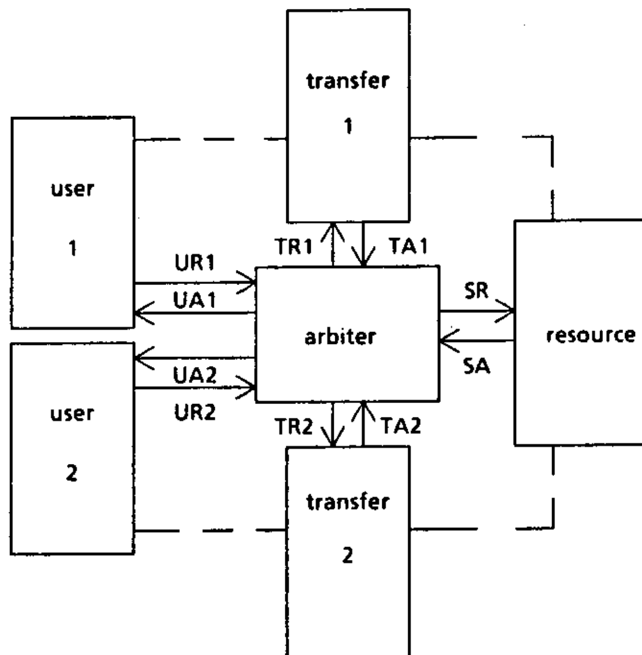


Figure 3.5 Arbiter and Associated Modules

The arbiter module shown in Figure 3.5 determines the order in which two requesting user modules obtain access to a shared resource. The various modules communicate via hand-shaking where requests are followed by acknowledgements according to a *four-cycle protocol*. The four cycles are:

(i) Requestor requests acknowledgement from responding unit.

(ii) Responder acknowledges request.

(iii) Requestor completes request.

(iv) Responder acknowledges completion of request.

Lines labelled xRi $(i=1,2)$ are request lines, and lines labelled xAi $(i=1,2)$ are acknowledgement lines.

The arbiter is the central control for the resource. Operation is as follows. After receiving requests from users the arbiter decides to which user it will grant access. It then invokes the corresponding transfer module to receive parameters to be used by the resource. The arbiter then request usage of the resource. After the resource acknowledges the request and the user is done using the resource, the cycle repeats.

Assumptions about the behavior of the user module may be expressed using the temporal until U and eventually ∇ operators. The safety assumption that the request lines for users Ui $(i=1,2)$ must stay at a Boolean value x until acknowledged by the arbiter is expressed

$$(URi=x) \supset (URi=x)\ U\ (UAi=x).$$

The liveness assumption that the user will eventually release the resource by pulling URi low is expressed

$$UAi \supset \nabla \neg URi.$$

Starting with several similar assumptions, a key invariant assertion is proven: after starting in initial state state 1, the arbiter will change state according to the state diagram of Figure 3.6 where the states labelled i, $i=2,\ldots,6$ are reached when user 1 is chosen, and the states labelled i', $i=2,\ldots,6$ are reached when user 2 is chosen. $UA1$ ($UA2$) is true in States 4, 5, and 6 (4', 5' and 6'). Since every cyclic path from state 1 returning to state 1 goes through either state 4, 5, or 6 (4', 5', or 6'), the desired property that an acknowledgement eventually follows a user request will hold.

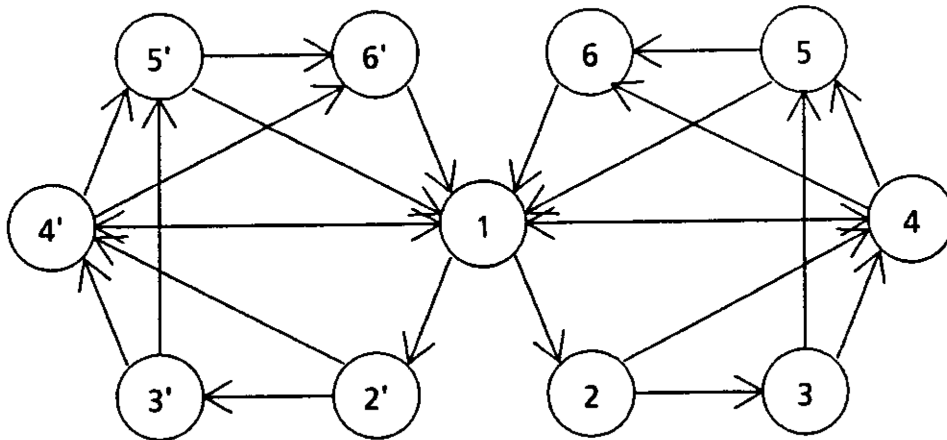


Figure 3.6 Internal Arbiter States

Bochmann assumes that gates within the arbiter module have zero delay. This has the advantage of restricting the number of possible states, but it would not be a valid assumption in asynchronous circuits whose correct operation depends upon the fact the gates have some non-zero delay.

The correctness proof itself requires knowing the intimate details of the arbiter, and this is to be expected. Since the proof is not done in a deductive system for temporal logic, it is not as formal as it could be. This would limit the ability to use proof checkers, and it may restrict the degree to which the proof could be automated.

3.3.4 Model Checker Approach

Clarke, Emerson, Mishra, and Dill [Mis83, Dil85] use a temporal logic model checker to check temporal specifications of a circuit on a finite state graph representation of the circuit's execution. An reachability algorithm is used to generate the graph

execution from the circuit schematic. Then the model checker is fed the temporal specifications and the graph and determines whether the graph satisfies the specifications. The method deals with gate-level circuits. Examples include the FIFO queue [Sei80a] discussed by Seitz using a unit gate delay assumption and the arbiter [Sei80b] using an unbounded delay assumption.

This approach does not involve proofs of correctness and the associated creativity in symbolic manipulation. It primarily involves a graph algorithm and, therefore, has been automated. It is not as formal as a correctness proof, however, because there is no guarantee that the graph is a correct representation of the circuit's execution. If an incorrect graph is produced, then the model checker may say that the temporal specifications are correct without a sound basis.

3.4 Relation to Our Work

Bochmann's arbiter proof was not done in a deductive system for temporal logic. This limits the degree to which a proof checker could be used. The model checker approach allows the possibility of incorrect models of execution to be considered. Both are gate-level approaches. In addition, neither approach has the functional specification property where specifications of complex modules are functions (transformations in the deductive system) of the specifications of the constituent parts. This limits the degree to which specifications of smaller modules can be used in proofs of larger modules containing the smaller modules.

We intend to focus on correctness proofs in the deductive system for PTL described in Chapter 2. Doing this provides the desired level of formalism and allows for proofs to be checked by a proof checker. The functional specification property is adhered to throughout, providing the basis for hierarchical verification. This is

demonstrated most fully in Chapter 6 where an asynchronous pipeline element is verified at the module level. Our Execution Graph Method of Chapter 6 employs the notion of a graph representing the circuit's execution as does the model checker approach. Our execution graphs, however, have a one-to-one correspondence to a set of temporal formulas which describe all allowable execution sequences. The formulas are proved from the axioms and, therefore, the graph is formally correct with respect to them.

CHAPTER 4

Steady-State Properties

Applying input forever allows us to focus on steady-state properties, in the meantime establishing the flavor of the TL deduction process. If one were to apply an input to a circuit forever and observe the corresponding response, the circuit would settle into some steady-state. This liveness is expressed

$$IC \ \& \ \Box Stimulus \supset \nabla \Box Response$$

where *IC* describes the initial conditions (initial circuit state), *Stimulus* describes some input stimulus to the circuit, and *Response* describes some state the circuit will remain in until further stimuli. We establish some proof rules (theorems of PTL) which allow us to construct proofs of safety and liveness properties. The first example circuit is a latch which is composed of two cross-coupled NAND gates. Then using the latch and the gate again as primitive elements, an asychronous controller composed of these devices is proved correct.

Since asynchronous circuits are primarily constructed out of gates in either combinational logic functions or feedback loops making up latches, gates will be the basic atomic component. A small proof system for proving the correctness of asynchronous

state machines will be presented. It will be used to prove the correctness of a controller taken from the literature.

4.1 A Proof System

The proof system begins with the axiomatization of temporal logic given in Chapter 3. We add tautologies of propositional calculus as theorems as was stated in inference rule R1, and we add theorems of temporal logic proven in [Man81], [Hai82], or this thesis in Section 4.1.8 below. This forms the domain-independent part of the system which is valid whether the objects to be verified are circuits or programs. The domain-dependent portion consists of axioms describing gates and proof rules which manipulate the axioms into desired assertions.

The proof approach is bottom-up. Axioms describe individual gates. As gates are composed to form circuits, the axioms are transformed via the proof rules and PTL formulas into the desired circuit specifications. We provide hand proofs of the proof rules to illustrate the type of manipulation required for proving theorems in the PTL semantics. The DP could be used instead of the hand proofs.

4.1 Steady-State Behavior

Consider an asynchronous state machine which is ordinarily in a stable state. It will react to input changes and settle again in a stable state. The timing of such a machine is shown in Figure 4.1. One interesting provable property of the machine is that it changes state correctly when the input changes. Rather than specifying exactly how long the machine is stable in terms of the O operator, we say that it is stable "forever" or "long enough." Thus we will use assertions of the form

$$\square \mathit{InputChange} \supset \nabla \square \mathit{CorrectStateChange}$$

where the $\square$ operator is used to state that the circuit is stable for a long time.

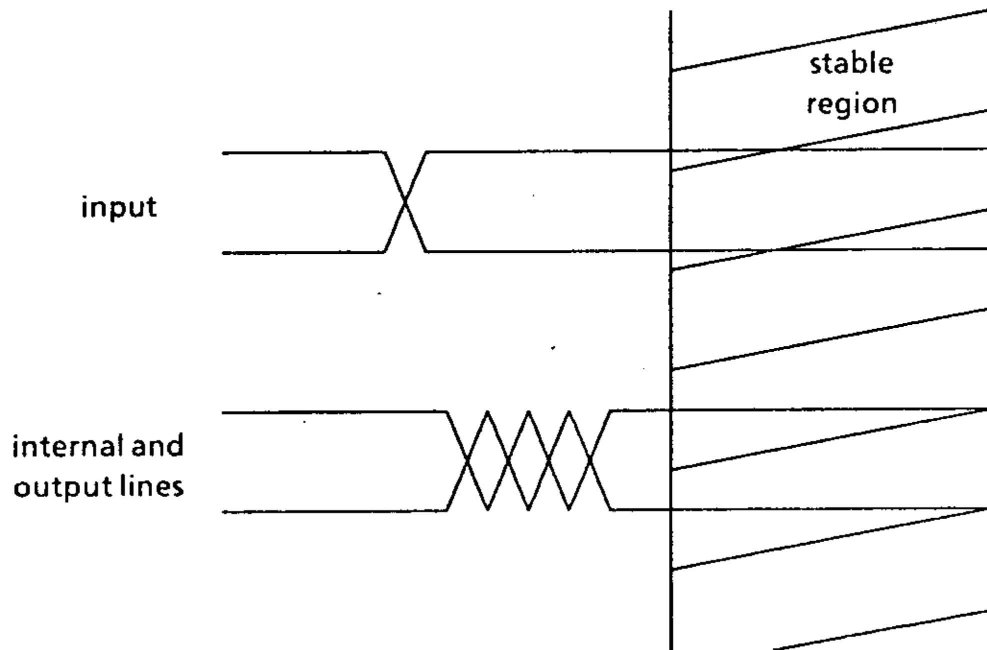


Figure 4.1 Timing of an Asynchronous Machine

4.1.2 Gate Properties

The output of a gate may be viewed as a time-delayed function of its input. As long as the inputs remain the same the output is constant. When the inputs change a new output value is determined after some finite delay. These properties are to be

expressed in the axioms of the proof system.

Let the box in Figure 4.2 represent a gate and β be the name of the output line. α is a propositional calculus (PC) formula which must be true for the gate to produce a 1 output and false for the gate to produce a 0 output. For a two-input NAND gate as in Figure 4.3, $\alpha \equiv \neg(I_1 \& I_2)$ and, when α is true, β (which is O) will be true at some time later. The two liveness axioms, G1 and G2, guarantee that the appropriate output change, namely β becoming true (false), will occur after an input change, α becoming true (false).

$$\alpha \twoheadrightarrow \beta \tag{G1}$$

$$\neg\alpha \twoheadrightarrow \neg\beta \tag{G2}$$

Notice that the axioms come in pairs which are duals of each other. These axioms state the expected gate property that, in the case of the NAND gate, when $\neg I_1 \vee \neg I_2$ is true then eventually O will become true and when $I_1 \& I_2$ is true then eventually $\neg O$ will become true.

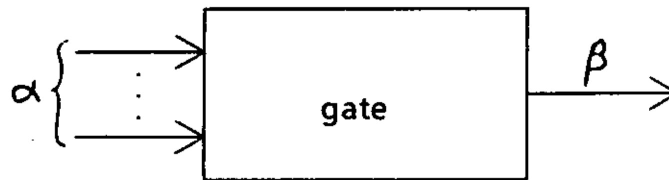


Figure 4.2 A Generalized Gate

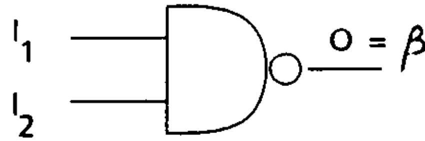


Figure 4.3 A NAND Gate

Gates retain the same output value as long as the inputs are constant. A safety axiom pair G3-G4 guarantees that the output does not change as long as the input does not change.

$$\beta \Rightarrow \beta \, U_A \, \neg \alpha \tag{G3}$$

$$\neg \beta \Rightarrow \neg \beta \, U_A \, \alpha \tag{G4}$$

Two more safety axioms form a restriction on the input behavior and the gate delay. They say that an input must be held constant long enough for the gate to react to the input.

$$\alpha \Rightarrow \alpha \, U_A \, \beta \tag{G5}$$

$$\neg \alpha \Rightarrow \neg \alpha \, U_A \, \neg \beta \tag{G6}$$

This axiom pair ensures that the gates obey a two-cycle protocol. One may think of the event when the input predicate α becomes true (false) as a request for the gate to switch. The event consisting of the output line β becoming true is then an acknowledgement. A two-cycle protocol guarantees that each request is answered by an

acknowledgement before another reqest can be made. Figure 4.4 shows α and β obeying a two-cycle protocol. If the input changes before the gate reacts to the previous change then these safety axioms are violated. These axioms are included to avoid one possible problem with the unbounded gate delay assumption: that the gate can take longer to react to input changes than input changes occur, causing a build-up in gate switching as shown in Figure 4.5. This would not model gate behavior accurately because the gate would appear to have a memory that would allow it to take its time in reacting to input changes.

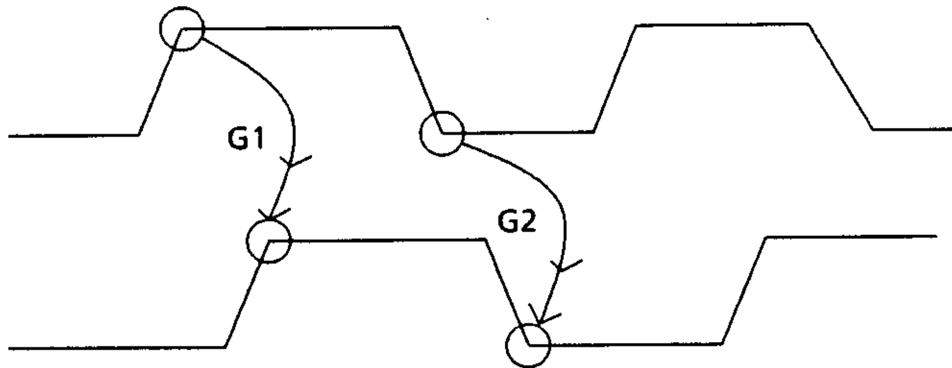


Figure 4.4 Liveness Properties Dictated by Axioms G1 and G2

Figure 4.4 shows what the liveness axioms (G1 and G2) dictate. The event of α becoming true (false) precedes and causes the event of β becoming true (false). Figure 4.6 shows what the four safety rules (G3 through G6) will allow.

The latch of Figure 4.7 is composed of two two-input NAND gates. Six axioms of the form G1 through G6 are needed to describe the behavior of each gate. For gate g1 the axioms are

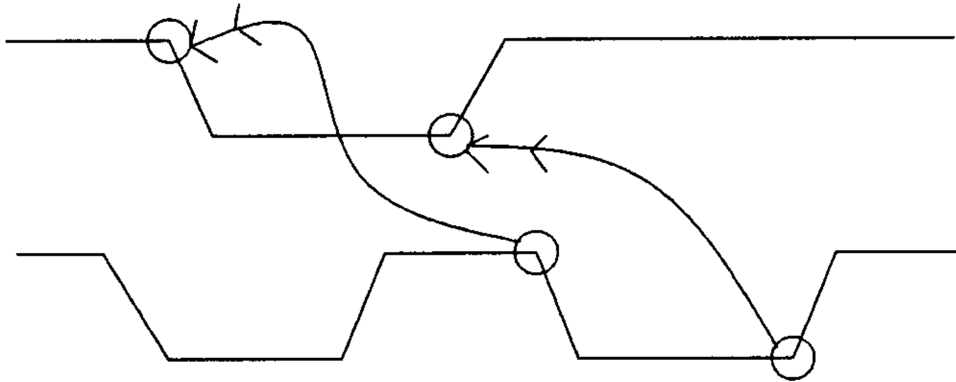


Figure 4.5 Physically Unrealizable Gate Switching

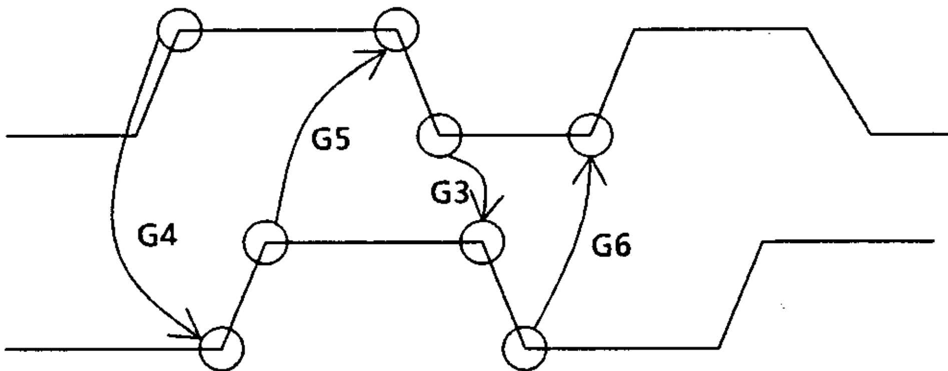


Figure 4.6 Switching Ordering Imposed by Safety Axioms

$$\vdash \bar{S} \,\&\, \bar{Q} \twoheadrightarrow \neg Q \qquad \text{(g1.G1)}$$

$$\vdash \neg \bar{S} \vee \neg \bar{Q} \twoheadrightarrow Q \qquad \text{(g1.G2)}$$

$$\vdash Q \Rightarrow Q \; U_A \; (\bar{S} \,\&\, \bar{Q}) \qquad \text{(g1.G3)}$$

$$\vdash \neg Q \Rightarrow \neg Q \; U_A \; (\neg \bar{S} \vee \neg \bar{Q}) \qquad \text{(g1.G4)}$$

$$\vdash (\bar{S} \;\&\; \bar{Q}) \Rightarrow (\bar{S} \;\&\; \bar{Q}) \; U_A \; \neg Q \qquad \text{(g1.G5)}$$

$$\vdash (\neg \bar{S} \vee \neg \bar{Q}) \Rightarrow (\neg \bar{S} \vee \neg \bar{Q}) \; U_A \; Q. \qquad \text{(g1.G6)}$$

For gate g2 the axioms are

$$\vdash Q \;\&\; \bar{R} \twoheadrightarrow \neg \bar{Q} \qquad \text{(g2.G1)}$$

$$\vdash \neg Q \vee \neg \bar{R} \twoheadrightarrow \bar{Q} \qquad \text{(g2.G2)}$$

$$\vdash \bar{Q} \Rightarrow \bar{Q} \; U_A \; (Q \;\&\; \bar{R}) \qquad \text{(g2.G3)}$$

$$\vdash \neg \bar{Q} \Rightarrow \neg \bar{Q} \; U_A \; (\neg Q \vee \neg \bar{R}) \qquad \text{(g2.G4)}$$

$$\vdash (Q \;\&\; \bar{R}) \Rightarrow (Q \;\&\; \bar{R}) \; U_A \; \neg \bar{Q} \qquad \text{(g2.G5)}$$

$$\vdash (\neg Q \vee \neg \bar{R}) \Rightarrow (\neg Q \vee \neg \bar{R}) \; U_A \; \bar{Q} \qquad \text{(g2.G6)}$$

Note that all the axioms use variations of the $\supset$ operator. The liveness axioms use the $\twoheadrightarrow$ operator defined

$$w_1 \twoheadrightarrow w_2 \equiv \Box (w_1 \supset \nabla w_2)$$

and the safety axioms use the $\Rightarrow$ operator defined

$$w_1 \Rightarrow w_2 \equiv \Box (w_1 \supset w_2).$$

This is actually unnecessary since the $\Box I$ rule will ensure that the axioms are valid for all time. The six axioms could be written

$$\vdash \alpha \supset \nabla \beta \qquad \text{(G1*)}$$

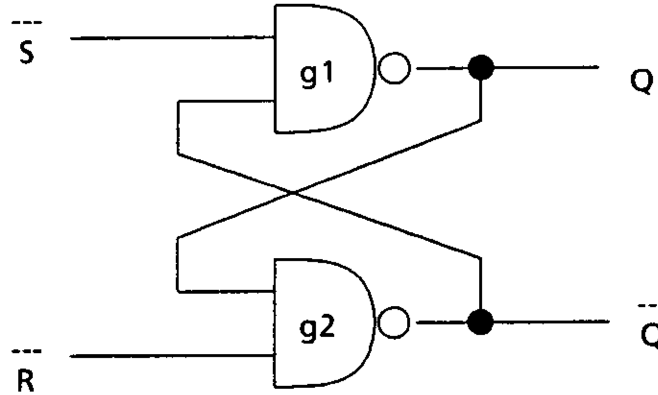


Figure 4.7 A Latch Composed of NAND Gates

$$\vdash \neg \alpha \supset \nabla \neg \beta \tag{G2*}$$

$$\vdash \beta \supset \beta\, U_A \neg \alpha \tag{G3*}$$

$$\vdash \neg \beta \supset \neg \beta\, U_A\, \alpha \tag{G4*}$$

$$\vdash \alpha \supset \alpha\, U_A\, \beta \tag{G5*}$$

$$\vdash \neg \alpha \supset \neg \alpha\, U_A \neg \beta \tag{G6*}$$

and rule $\Box I$ applied to obtain the other forms (G1-G6) when needed.

When axiomatizing gate behavior and proving properties of the asynchronous machines which the gates are composed of, the assumption of unbounded gate delay has quite an impact. An alternative assumption of one-unit gate delays can lead to simpler axioms and a simpler analysis but is less realistic [Mis83]. Under the one-unit

delay assumption, the axioms needed to characterize gate behavior are

$$\vdash \alpha \Rightarrow \bigcirc\beta \qquad \text{(G1')}$$

$$\vdash \neg\alpha \Rightarrow \bigcirc\neg\beta \qquad \text{(G2')}$$

$$\vdash \beta \Rightarrow \beta\, U_A\, \neg\alpha \qquad \text{(G3')}$$

$$\vdash \neg\beta \Rightarrow \neg\beta\, U_A\, \alpha. \qquad \text{(G4')}$$

The first two axioms replace the G1 and G2 liveness axioms. Rather than having the output change "eventually," G1' and G2' have the output change after exactly one time unit. G3' and G4' are the same as G3 and G4. The property of holding the output value as long as the inputs remain constant is applicable under both assumptions. No counterpart to G5 and G6 is needed because the output changes keep up with the input changes. Since unbounded gate delays will be assumed G1 through G6, rather than G1' through G4', will be used from now on.

4.1.3 Input Elimination Rules

The input elimination rules are used to simplify gate axioms, eliminating the effect of input lines when they are at a certain stable value forever. For example if a two-input NAND gate always has a high-valued input, that input may effectively be ignored and the NAND gate behaves like an inverter. A NOR gate becomes an inverter if one of the inputs is low forever. Output safety axioms like G3 and G4, which say that the output of a gate does not change value unless the input has changed, may be simplified by the IEOS (Input Elimination Output Safety) inference rules.

$$\frac{\vdash w \supset \Box\neg\alpha_1 \quad , \quad \vdash w \supset \beta \Rightarrow \beta\, U_A\, (\alpha_1 \vee \alpha_2)}{\vdash w \supset \beta \Rightarrow \beta\, U_A\, \alpha_2} \qquad \text{(IEOS1)}$$

$$\frac{\vdash w \supset \Box \alpha_1 \quad , \quad \vdash w \supset \beta \Rightarrow \beta\, U_A\, (\alpha_1 \,\&\, \alpha_2)}{\vdash w \supset \beta \Rightarrow \beta\, U_A\, \alpha_2} \qquad \text{(IEOS2)}$$

where α_1, α_2, and β are PC formulas.

These rules are used to reduce the dependence of the output line β on the inputs α_1 and α_2, that is, to eliminate α_1 (or α_2 by commutativity). The rules may be proved valid by

Theorem 4.1. The implications

$$\vdash [\Box \neg \alpha_1] \;\&\; [\beta \Rightarrow \beta\, U_A\, (\alpha_1 \vee \alpha_2)] \;\supset\; [\beta \Rightarrow \beta\, U_A\, \alpha_2] \qquad \text{(IEOSimp1)}$$

and

$$\vdash [\Box \alpha_1] \;\&\; [\beta \Rightarrow \beta\, U_A\, (\alpha_1 \,\&\, \alpha_2)] \;\supset\; [\beta \Rightarrow \beta\, U_A\, \alpha_2]. \qquad \text{(IEOSimp2)}$$

are valid.

Theorem 4.1 is proved in Appendix A.

To establish the inference rules, use the PC tautology

$$(u_1 \supset u_2) \;\supset\; ((w \supset u_1) \supset (w \supset u_2)) \qquad \text{(PC1)}$$

to introduce w into each side of the implications. Applying the PC tautology

$$w \supset (u_1 \,\&\, u_2) \equiv (w \supset u_1) \,\&\, (w \supset u_2) \qquad \text{(PC2)}$$

to the antecedent of each of the implications and applying the rule of propositional reasoning PR establishes the inference rules.

Input safety axioms which take the form of G5 and G6 may be simplified using the Input Elimination Input Safety (IEIS) rules

$$\frac{\vdash w \supset \Box \neg \alpha_1 \quad , \quad \vdash w \supset \alpha_1 \vee \alpha_2 \Rightarrow \alpha_1 \vee \alpha_2 \, U_A \, \beta}{\vdash w \supset \alpha_1 \vee \alpha_2 \Rightarrow \alpha_2 \, U_A \, \beta} \qquad \text{(IEIS1)}$$

and

$$\frac{\vdash w \supset \Box \alpha_1 \quad , \quad \vdash w \supset \alpha_1 \,\&\, \alpha_2 \Rightarrow \alpha_1 \,\&\, \alpha_2 \, U_A \, \beta}{\vdash w \supset \alpha_1 \,\&\, \alpha_2 \Rightarrow \alpha_2 \, U_A \, \beta} \qquad \text{(IEIS2)}$$

The IEIS rules are proven valid by

Theorem 4.2. The implications

$$\vdash [\Box \neg \alpha_1] \;\&\; [\alpha_1 \vee \alpha_2 \Rightarrow \alpha_1 \vee \alpha_2 \, U_A \, \beta] \;\supset \qquad \text{(IEISimp1)}$$

$$[\alpha_1 \vee \alpha_2 \Rightarrow \alpha_2 \, U_A \, \beta]$$

and

$$\vdash [\Box \alpha_1] \;\&\; [\alpha_1 \,\&\, \alpha_2 \Rightarrow \alpha_1 \,\&\, \alpha_2 \, U_A \, \beta] \;\supset \qquad \text{(IEISimp2)}$$

$$[\alpha_1 \,\&\, \alpha_2 \Rightarrow \alpha_2 \, U_A \, \beta]$$

are valid.

Theorem 4.2 is proved in Appendix A.

The PC tautologies PC1 and PC2 are used to introduce the precondition w and

propositional reasoning establishes the inference rules as above for the IEOS rules.

4.1.4 Stability Implication Rule

The Stability Implication (SI) rule allows one to deduce that henceforth the output of a gate will not change given that henceforth the input does not change. The output safety axioms (G3 and G4) can be simplified, eliminating the U_A operator. The SI rule is stated

$$\frac{\vdash w \supset \Box\alpha \ , \ \vdash w \supset \beta \Rightarrow \beta\, U_A\, (\neg\alpha)}{\vdash w \supset \beta \Rightarrow \Box\beta} \qquad \text{(SI)}$$

It can be proved valid by proving

Theorem 4.3. The implication

$$\vdash [\Box\alpha] \ \& \ [\beta \Rightarrow \beta\, U_A\, (\neg\alpha)] \supset [\beta \Rightarrow \Box\beta] \qquad \text{(SIimp)}$$

is valid.

Theorem 4.3 is proved in Appendix A.

The PC tautologies PC1 and PC2 are used to introduce the precondition w and propositional reasoning establishes the inference rules as above for the IEOS rules.

4.1.5 Inverter-Inverter Stability Rule

When a pair of gate output safety axioms have been reduced by input elimination to where only one significant input remains in each axiom, the Inverter-Inverter Stability (IIS) rule may be applied to deduce that the gate output will remain stable forever. The rule gets its name from the fact that it models two inverters in a feedback loop.

When two cross-coupled inverters have line values (high or low) they retain those values forever since there are no other inputs to upset the stability (please see Figure 4.8). When it is desirable to show that a latch composed of cross-coupled gates will be stable, this rule can be used. The IIS rule is

$$\vdash w \supset \beta_1 \tag{IIS}$$

$$\vdash w \supset \beta_2$$

$$\vdash w \supset \beta_1 \Rightarrow \beta_1 \, U_A \, (\neg \beta_2)$$

$$\frac{\vdash w \supset \beta_2 \Rightarrow \beta_2 \, U_A \, (\neg \beta_1)}{\vdash w \supset \Box(\beta_1 \,\&\, \beta_2)}$$

and can be proved valid by showing

Theorem 4.4. The implication

$$\vdash \beta_1 \;\&\; \beta_2 \;\&\; [\beta_1 \Rightarrow \beta_1 \, U_A \, (\neg \beta_2)] \;\& \tag{4.9}$$

$$[\beta_2 \Rightarrow \beta_2 \, U_A \, (\neg \beta_1)] \supset \Box(\beta_1 \,\&\, \beta_2)$$

is valid.

Theorem 4.4 is proved in Appendix A.

The PC tautologies PC1 and PC2 are used to introduce the precondition w and propositional reasoning establishes the IIS inference rule from formula (4.9) as above for the IEOS rules.

4.1.6 Inverter-Gate Stability Rule

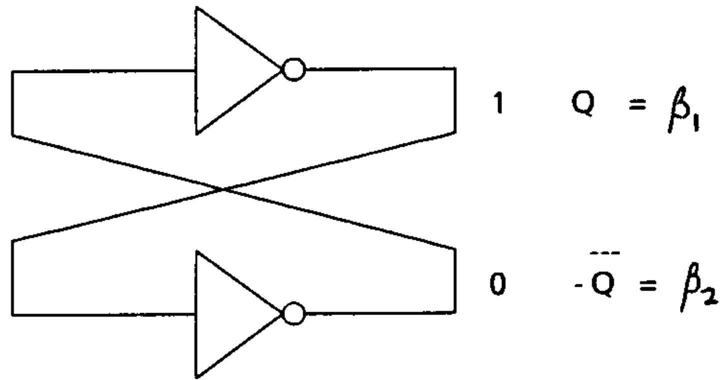


Figure 4.8 Cross-coupled Inverters

The IIS rule of Section 4.1.5 allows one to deduce that cross-coupled gates reduced by input elimination to cross-coupled inverters are stable. Another situation arises when one gate has been reduced by input elimination to an inverter and the other cannot be reduced, but the circuit is stable. This situation is shown in Figure 4.9. The circuit in the figure is a NAND gate latch with the bottom NAND gate reduced to an inverter by the assumption $\Box \bar{R}$. Now the $\bar{S}$ line can be high or low without affecting the latch state since it is already set. The goal is to be able to deduce that the $\bar{S}$ input is insignificant and the circuit is stable. The Inverter-Gate Stability (IGS) rule will be introduced for this purpose. The IGS rule is

$$\vdash w \supset \beta_1 \,\&\, \beta_2 \qquad \text{(IGS)}$$

$$\vdash w \supset \beta_1 \Rightarrow \beta_1 \, U_A \, (\neg \beta_2)$$

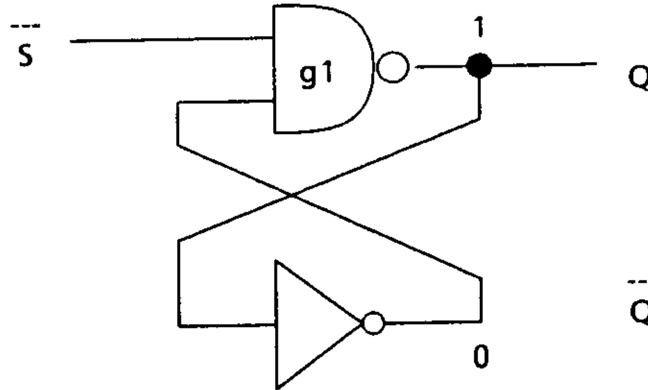


Figure 4.9 A Stable Latch in the Set State

$$\frac{\vdash w \supset \beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1 \,\&\, \alpha)}{\vdash w \supset \Box(\beta_1 \,\&\, \beta_2)}$$

The IGS rule may be proved valid by

Theorem 4.5. The implication

$$\vdash \beta_1 \,\&\, \beta_2 \,\&\, [\beta_1 \Rightarrow \beta_1\, U_A\, (\neg \beta_2)] \,\&$$

$$[\beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1 \,\&\, \alpha)] \supset \Box(\beta_1 \,\&\, \beta_2)$$

is valid.

Theorem 4.5 is proved in Appendix A.

The PC tautologies PC1 and PC2 are used to introduce the precondition w and propositional reasoning establishes the IGS inference rule as above for the IEOS rules.

A variation of the IGS rule is used when only the weaker condition $\nabla(\beta_1 \,\&\, \beta_2)$ is known. This variation is called the Eventual Inverter-Gate Stability Rule (EIGS).

$$\begin{array}{l} \vdash w \supset \nabla(\beta_1 \,\&\, \beta_2) \\ \vdash w \supset \beta_1 \Rightarrow \beta_1\, U_A\, (\neg \beta_2) \\ \vdash w \supset \beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1 \,\&\, \alpha) \\ \hline \vdash w \supset \nabla \Box (\beta_1 \,\&\, \beta_2) \end{array} \qquad \text{(EIGS)}$$

The associated implication is

$$\nabla(\beta_1 \,\&\, \beta_2) \;\&\; [\beta_1 \Rightarrow \beta_1\, U_A\, (\neg \beta_2)] \;\& \qquad \text{(EIGSimp)}$$
$$[\beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1 \,\&\, \alpha)] \supset \nabla \Box (\beta_1 \,\&\, \beta_2).$$

EIGSimp can be proved within the PTL deductive system.

Theorem 4.6. The implication

$$\nabla(\beta_1 \,\&\, \beta_2) \,\&\, [\beta_1 \Rightarrow \beta_1\, U_A\, (\neg \beta_2)] \;\&$$
$$[\beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1 \,\&\, \alpha)] \supset \nabla \Box (\beta_1 \,\&\, \beta_2)$$

is valid.

Theorem 4.6 is proved in Appendix A.

4.1.7 Coincidence Rule

The Coincidence Rule (CR) is used to deduce that at some point in time both α and β are true if $\alpha\, U_A\, \beta$ and $\nabla \beta$ are assumed. The rule is stated

$$\frac{\vdash \alpha\, U_A\, \beta\ ,\ \vdash \nabla \beta}{\vdash \nabla (\alpha \,\&\, \beta)} \qquad \text{(CR)}$$

and can be justified by

Theorem 4.7. The implication

$$\vdash ((\alpha\, U_A\, \beta) \,\&\, \nabla \beta) \supset \nabla (\alpha \,\&\, \beta) \qquad \text{(CRimp)}$$

is valid.

Theorem 4.7 is proved in Appendix A.

4.1.8 Additional Theorems

The following theorems are useful when doing correctness proofs. They will be used in proving a controller correct.

Theorem $T32$. $\vdash (u \supset \Box(v \supset w_1)) \,\&\, (u \supset \Box(v \supset w_2)) \equiv$

$$u \supset \Box(v \supset (w_1 \,\&\, w_2))$$

Proof

(1) $\vdash (u \supset \Box(v \supset w_1) \,\&\, (u \supset \Box(v \supset w_2))$

$\equiv u \supset (\Box(v \supset w_1) \,\&\, \Box(v \supset w_2))$ by the PC tautology

$(w \supset u) \,\&\, (w \supset v) \equiv$

$w \supset (u \,\&\, v)$

(2) $\vdash u \supset (\Box(v \supset w_1) \,\&\, \Box(v \supset w_2)) \equiv$

$u \supset \Box((v \supset w_1) \& (v \supset w_2))$ by T4

(3) $\vdash u \supset \Box((v \supset w_1) \& (v \supset w_2)) \equiv$

$u \supset \Box(v \supset (w_1 \& w_2))$ by the PC tautology

$(w \supset u) \& (w \supset v) \equiv$

$w \supset (u \& v)$

(4) $\vdash (u \supset \Box(v \supset w_1)) \& (u \supset \Box(v \supset w_2)) \equiv$

$u \supset \Box(v \supset (w_1 \& w_2))$ by $\supset$ Trans. twice using

(1),(2),(3)

•

Theorem $T33$. $\vdash \nabla w_1 \& \Box w_2 \supset \nabla(w_1 \& \Box w_2)$

Proof

(1) $\vdash \nabla w_1 \& \Box w_2 \equiv \nabla w_1 \& \Box(\Box w_2)$ by T1 and ER

(2) $\vdash \nabla w_1 \& \Box(\Box w_2) \supset \nabla(w_1 \& \Box w_2)$ by T7

(3) $\vdash \nabla w_1 \& \Box w_2 \supset \nabla(w_1 \& \Box w_2)$ by $\supset$ Trans. using (1),(2)

•

Theorem $T34$. $\vdash (\Box w_1 \& \nabla \Box w_2) \supset \nabla \Box(w_1 \& w_2)$

Proof

(1) $\vdash w \supset \nabla w$ by T31

(2) $\vdash \Box w_1 \& \ \nabla \Box w_2 \supset$
$\nabla \Box w_1 \& \ \nabla \Box w_2$ by the PC tautology
$(w_1 \supset w_2) \supset (w_1 \& \ u \supset w_2 \& \ u)$
using (1) with $\nabla \Box w_2 \equiv u$

(3) $\vdash \nabla \Box w_1 \& \ \nabla \Box w_2 \supset$
$\nabla \Box (w_1 \& \ w_2)$ by T8

(4) $\vdash \Box w_1 \& \ \nabla \Box w_2 \supset \nabla \Box (w_1 \& \ w_2)$ by $\supset$ Trans. using (2),(3)

•

4.2 Proving Safety and Liveness

The proof system will be used to verify asynchronous circuits. First we consider properties of latches constructed out of gates. Then we verify a controller composed of gates and latches.

4.2.1 Latch Properties

The $\bar{S}-\bar{R}$ latch is a basic memory element in asynchronous circuits. The latch of Figure 4.7 has two active high inputs and no clock. A few predicates will be used for abbreviation.

$ResetInput \equiv \neg \bar{R}$

$SetInput \equiv \neg \bar{S}$

$ResetState \equiv \neg Q \ \& \ \bar{Q}$

$SetState \equiv Q \ \& \ \neg \bar{Q}$

The latch possesses *static properties*, which are assertions about its behavior with no input change. These are defined

$$LatchStayReset \equiv ResetState \ \& \ \Box \neg SetInput \supset \Box ResetState$$

$$LatchStaySet \equiv SetState \ \& \ \Box \neg ResetInput \supset \Box SetState.$$

LatchReset can be proved from the basic gate axioms as follows.

Let $L_0 \equiv ResetState \ \& \ \Box \neg SetInput \equiv (\neg Q \ \& \ \bar{Q}) \ \& \ \Box \bar{S}$. Then the deduction of *LatchStayReset* proceeds as follows.

(1)	$\vdash L_0 \supset \Box \bar{S}$	
(2)	$\vdash \neg Q \Rightarrow \neg Q \ U_A \ (\neg \bar{S} \vee \neg \bar{Q})$	by axiom g1.G4
(3)	$\vdash L_0 \supset [\neg Q \Rightarrow \neg Q \ U_A \ (\neg \bar{S} \vee \neg \bar{Q})]$	by the PC tautology $w \supset (v \supset w)$
(4)	$\vdash L_0 \supset [\neg Q \Rightarrow \neg Q \ U_A \ \neg \bar{Q}]$	by IEOS1 using (1),(3)
(5)	$\vdash \bar{Q} \Rightarrow \bar{Q} \ U_A \ (Q \ \& \ \bar{R})]$	by axiom g2.G3
(6)	$\vdash L_0 \supset [\bar{Q} \Rightarrow \bar{Q} \ U_A \ (Q \ \& \ \bar{R})]$	by the PC tautology $w \supset (v \supset w)$
(7)	$\vdash L_0 \supset \neg Q$	
(8)	$\vdash L_0 \supset \bar{Q}$	
(9)	$\vdash L_0 \supset \Box(\neg Q \ \& \ \bar{Q})$	by IGS using (7),(8),(4),(6)

The dual property *LatchStaySet* is proved similarly.

The latch also possesses *dynamic properties*, which describe its dynamic behavior when one of the inputs change. These are defined

$$LatchReset \equiv \Box \neg ResetInput \;\&\; \nabla SetInput \supset \nabla \Box SetState$$

$$LatchSet \equiv \Box \neg ResetInput \;\&\; \nabla SetInput \supset \nabla \Box SetState.$$

LatchSet says that if the reset input is never applied and the set input is applied at some time then eventually the latch will be set and will continue to be set forever. The proof proceeds as follows.

(1) $\vdash \nabla \neg \overline{S} \supset \nabla \neg \overline{S} \vee \nabla \neg \overline{Q}$	by the PC tautology $u \supset u \vee v$
(2) $\vdash \nabla \neg \overline{S} \vee \nabla \neg \overline{Q} \supset \nabla (\neg \overline{S} \vee \neg \overline{Q})$	by T5
(3) $\vdash \nabla \neg \overline{S} \supset \nabla (\neg \overline{S} \vee \neg \overline{Q})$	by $\supset$ Trans. using (1),(2)
(4) $\vdash \Box (\neg \overline{S} \vee \neg \overline{Q} \supset \nabla Q)$	by axiom g1.G6
(5) $\vdash \nabla (\neg \overline{S} \vee \neg \overline{Q}) \supset \nabla \nabla Q$	by T3
(6) $\vdash \nabla (\neg \overline{S} \vee \neg \overline{Q}) \supset \nabla Q$	by T2 and ER
(7) $\vdash \nabla \neg \overline{S} \supset \nabla Q$	by $\supset$ Trans. using (3),(6)
(8) $\vdash \Box \overline{R} \;\&\; \nabla \neg \overline{S} \supset \Box \overline{R} \;\&\; \nabla Q$	by the PC tautology $(w_1 \supset w_2) \supset (u \;\&\; w_1 \supset u \;\&\; w_2)$
(9) $\vdash \Box \overline{R} \;\&\; \nabla Q \supset \nabla (\overline{R} \;\&\; Q)$	by T7
(10) $\vdash \Box \overline{R} \;\&\; \nabla \neg \overline{S} \supset \nabla (Q \;\&\; \overline{R})$	by $\supset$ Trans. using (8),(9)

Now the subgoal $\vdash \Box \overline{R} \;\&\; \nabla \neg \overline{S} \supset \nabla (Q \;\&\; \overline{R})$ has been achieved. Intuitively this is saying that, with $\Box \overline{R} \;\&\; \nabla \neg \overline{S}$ as an initial precondition, Q will go high (line 7) and eventually both inputs to gate g2 will be high simultaneously (line 10).

(11) $\vdash \Box\bar{R}\ \&\ \nabla\neg\bar{S}\ \supset\ \Box\bar{R}$	by the PC tautology $u\ \&\ v \supset u$
(12) $\vdash Q\ \&\ \bar{R} \Rightarrow Q\ \&\ \bar{R}\ U_A \neg\bar{Q}$	by axiom g2.G5
(13) $\vdash (\Box\bar{R}\ \&\ \nabla\neg\bar{S})\ \supset$ $[Q\ \&\ \bar{R} \Rightarrow Q\ \&\ \bar{R}\ U_A \neg\bar{Q}]$	by the PC tautology $w \supset (v \supset w)$
(14) $\vdash (\Box\bar{R}\ \&\ \nabla\neg\bar{S})\ \supset$ $[Q\ \&\ \bar{R} \Rightarrow Q\ U_A \neg\bar{Q}]$	by the rule IEIS2 using (11),(13)
(15) $\vdash Q\ \&\ \bar{R} \Rightarrow \neg\bar{Q}$	by axiom g2.G1
(16) $\vdash \Box\bar{R}\ \&\ \nabla\neg\bar{S}\ \supset\ [Q\ \&\ \bar{R} \Rightarrow \nabla\neg\bar{Q}]$	by the PC tautology $w \supset (v \supset w)$
(17) $\vdash \Box\bar{R}\ \&\ \nabla\neg\bar{S}\ \supset\ [Q\ \&\ \bar{R} \Rightarrow (Q\ U_A \neg\bar{Q})$ $\&\ \Box\bar{R}\ \&\ \nabla\neg\bar{S} \supset [Q\ \&\ \bar{R} \Rightarrow \nabla\neg\bar{Q}]$	by $(\vdash u)\ \&\ (\vdash v) \supset$ $\vdash u\ \&\ v$ of Pred. Calc.
(18) $\vdash \Box\bar{R}\ \&\ \nabla\neg\bar{S}\ \supset$ $[Q\ \&\ \bar{R} \Rightarrow (Q\ U_A \neg\bar{Q})\ \&\ \nabla\neg\bar{Q}]$	by T32 using (17)
(19) $\vdash [Q\ \&\ \bar{R} \Rightarrow (Q\ U_A \neg\bar{Q})\ \&\ \nabla\neg\bar{Q}]$ $\supset\ [\nabla(Q\ \&\ \bar{R}) \supset$ $\nabla((Q\ U_A \neg\bar{Q})\ \&\ \nabla\neg\bar{Q})]$	by T3
(20) $\vdash \Box\bar{R}\ \&\ \nabla\neg\bar{S}\ \supset$ $[\nabla(Q\ \&\ \bar{R}) \supset$ $\nabla((Q\ U_A \neg\bar{Q})\ \&\ \nabla\neg\bar{Q})]$	by $\supset$ Trans.

using (18),(19)

(21) $\vdash \nabla (Q \,\&\, \bar{R}) \supset$

$[\Box \bar{R} \,\&\, \nabla \neg \bar{S} \supset$

$\nabla ((Q\ U_A \neg \bar{Q}) \,\&\, \nabla \neg \bar{Q})]$ — by the PC tautology

$u \supset (v \supset w) \equiv$

$v \supset (u \supset w)$

(22) $\vdash \Box \bar{R} \,\&\, \nabla \neg \bar{S} \supset$

$[\Box \bar{R} \,\&\, \nabla \neg \bar{S} \supset$

$\nabla ((Q\ U_A \neg \bar{Q}) \,\&\, \nabla \neg \bar{Q})]$ — by $\supset$ Trans.

using (10),(21)

(23) $\vdash \Box \bar{R} \,\&\, \nabla \neg \bar{S} \supset \nabla$

$((Q\ U_A \neg \bar{Q}) \,\&\, \nabla \neg \bar{Q})$ — by the PC tautology

$u \supset (u \supset v) \equiv u \supset v$

(24) $\vdash (Q\ U_A \neg \bar{Q} \,\&\,) \nabla \neg \bar{Q} \supset$

$\nabla (Q \,\&\, \neg \bar{Q})$ — by CRimp

(25) $\vdash \nabla ((Q\ U_A \neg \bar{Q}) \,\&\, \nabla \neg \bar{Q}) \supset$

$\nabla\ \nabla (Q \,\&\, \neg \bar{Q})$ — by $\nabla\ \nabla$ Rule using (24)

(26) $\vdash \nabla ((Q\ U_A \neg \bar{Q}) \,\&\, \nabla \neg \bar{Q}) \supset$

$\nabla (Q \,\&\, \neg \bar{Q})$ — by T2

(27) $\vdash \Box \bar{R} \,\&\, \nabla \neg \bar{S} \supset \nabla (Q \,\&\, \neg \bar{Q})$ — by $\supset$ Trans.

using (23),(26)

(28) $\vdash \neg \bar{Q} \Rightarrow \neg \bar{Q}\ U_A (\neg Q \vee \neg \bar{R})$ — by axiom g2.G4

(29) $\vdash (\Box \bar{R} \,\&\, \nabla \neg \bar{S}) \supset$

$[\neg \bar{Q} \Rightarrow \neg \bar{Q}\ U_A \neg (\neg Q \vee \neg \bar{R})]$ by the PC tautology

$w \supset (v \supset w)$

(30) $\vdash (\Box \bar{R}\ \&\ \nabla \neg \bar{S}) \supset$

$[\neg \bar{Q} \Rightarrow \neg \bar{Q}\ U_A \neg Q]$ by the rule IEOS1

using (11),(29)

(31) $\vdash Q \Rightarrow Q\ U_A\ (\bar{S}\ \&\ \bar{Q})$ by axiom (g1.G3)

(32) $\vdash (\Box \bar{R}\ \&\ \nabla \neg \bar{S}) \supset$

$[Q \Rightarrow Q\ U_A\ (\bar{S}\ \&\ \bar{Q})]$ by the PC tautology

$w \supset (v \supset w)$

Lines 27, 29, and 32 are almost what is needed for the Eventual Inverter-Gate Stability inference rule (EIGS). Three more steps will yield the required antecedent.

(34) $\vdash \Box \bar{R}\ \&\ \nabla \neg \bar{S} \supset \nabla (Q\ \&\ \neg \bar{Q})$

$\&\ [\neg \bar{Q} \Rightarrow \neg \bar{Q}\ U_A \neg Q]\ \&$

$[Q \Rightarrow Q\ U_A\ (\bar{S}\ \&\ \bar{Q})]$ by the PC tautology

$(w \supset u)\ \&\ (w \supset v) \equiv$

$w \supset (u\ \&\ v)$

applied twice

using (27),(29),(32)

(35) $\vdash \nabla (Q\ \&\ \neg \bar{Q})\ \&\ [\neg \bar{Q} \Rightarrow \neg \bar{Q}\ U_A \neg Q]$

$\&\ [Q \Rightarrow Q\ U_A\ (\bar{S}\ \&\ \bar{Q})] \supset \nabla (Q\ \&\ \neg \bar{Q}$

$\&\ [\neg Q \Rightarrow \neg \bar{Q}\ U_A \neg Q]\ \&$

$[Q \Rightarrow Q\ U_A\ (\bar{S}\ \&\ \bar{Q})])$ by T33

(36) $\vdash \Box \bar{R}\ \&\ \nabla \neg \bar{S} \supset \nabla (Q\ \&\ \neg \bar{Q}$

$\& \; [\neg Q \Rightarrow \neg \bar{Q} \; U_A \; \neg Q] \; \&$

$[Q \Rightarrow Q \; U_A \; (\bar{S} \; \& \; \bar{Q})])$ by $\supset$ Trans. using (34),(35)

Applying EIGS to the line 36 yields the goal assertion that eventually the latch will be set after toggling the set input.

(37) $\vdash (\Box \bar{R} \; \& \; \nabla \neg \bar{S}) \; \supset \; \nabla \; \Box (Q \; \& \; \neg \bar{Q})$

This establishes the latch property LatchSet. Its dual, LatchReset is established similarly.

4.2.2 Correctness of an Asynchronous Controller

Consider an asynchronous controller which accepts inputs, guiding it through a cyclic sequence of states. If the controller is in state q_i at the present time and input p_{i+1} occurs, then it enters the state q_{i+1} (where all addition (+) is done modulo n, the number of states). The machine is assumed to have output lines which are combinational functions of the state.

The controller implements the automaton M of Figure 4.10 which is defined as the 5-tuple

$M = (\mathrm{Q},\mathrm{P},\delta,q_0,\mathrm{Q})$ *where* Q *is the state set* $\{q_0, \cdots ,q_{n-1}\}$

and P *is the set of input symbols* $\{p_0, \cdots ,p_{n-1}\}$

and δ *is the transition function defined* $\delta(q_i ,p_{i+1}) = q_{i+1}$

and $\delta(q_i ,p_j) = q_i, \; \forall j \neq i+1$

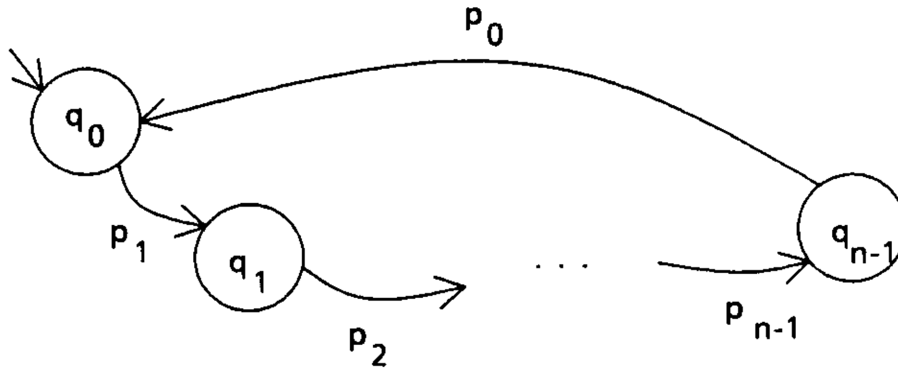


Figure 4.10 An Asynchronous State Machine

and q_0 is the initial state

and **Q** *is the set of final states.*

The machine, implemented with a *one-hot* state assignment [Hol82] appears in Figure 4.11. With a one-hot state assignment one latch has a high value, the latch corresponding to the state, and all others are low. If the machine is in state q_0 where Q_0 is high, for example, and input P_1 goes high then the machine changes state to q_1. If any other input P_j is applied with Q_0 high, then no state change occurs.

Operation of the machine is as follows. When the machine is in a settled state q_i and no input is applied, Q_i is high and all other Q_j's are low as expressed by the assertion

$$InState(q_i) \equiv SetState(i) \ \& \bigwedge_{j \neq i} ResetState(j).$$

In addition, all the P_i's are low. Since the P_i's are input to the NAND gates, all the latch set lines $\overline{S_i}$ will be high as expressed by

$$SetLinesSettled \equiv \bigwedge_{k=0}^{n-1} \overline{S_k}.$$

When any input P_j other than P_{i+1} is applied, it has no effect because of the low Q_{j-1} line feeding into the NAND gate j. If P_{i+1} is applied, however, the high Q_i line enables the NAND gate $i+1$ and the latch set input is activated, setting latch $i+1$. When the latch is set, then the $\overline{Q}_{i+1}$ output will go low, resetting latch i. Thus, after settling, the state q_{i+1} will be reached. Now pulling P_{i+1} low has no effect because NAND gate $i+1$ is disabled by the output line Q_i of latch i.

Correct operation of the machine is expressed by the temporal logic predicate *StateChange* defined

$$StateChange \equiv \forall q_i \in \mathbf{Q}.(InState(q_i) \ \& \ Input(p_{i+1}) \ \& \ SetLinesSettled$$

$$\twoheadrightarrow \ \Box InState(q_{i+1}))$$

where

$$Input(p_k) \equiv (\Box P_k) \ \& \bigwedge_{j \neq k} \Box \neg P_j.$$

StateChange will be proved by proving some intermediate properties of smaller portions of the machine. The latches and latch properties where already considered in the preceding section. That part of the machine which changes during a state change, hereafter called the *dynamic portion*, is first considered. This is the pair of latches and associated NAND gates for the current state and the next state (as explained in the discussion of the machine's operation. Then it will be shown that the rest of the latches

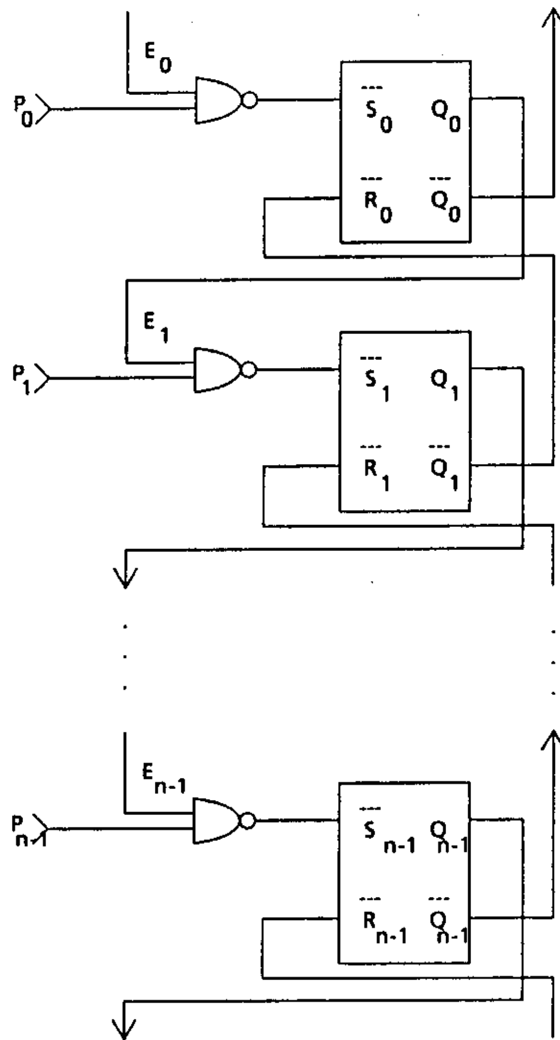


Figure 4.11 One-Hot Implementation of the Controller

are not affected by the state change. After all of the intermediate properties are established, *StateChange* follows naturally. The precondition for changing state

$$InState\,(q_i)\ \&\ Input\,(p_{i+1})\ \&\ SetLinesSettled$$

will be abbreviated A_0. Correctness of the machine will be established relative to this precondition.

When establishing properties of the dynamic portion of the machine, a weaker precondition implied by A_0 called A_1 will be used.

$$A_1 \equiv SetState(i) \ \& \ ResetState(i+1) \ \& \ \Box \neg SetInput(i) \ \& \ \Box \neg ResetInput(i+1) \ \& \ \Box P_{i+1}$$

The proof begins by dealing with the NAND gate associated with latch $(i+1)$. The latch properties *LatchReset* and *LatchSet* are used in the proof. The particular instances of these latch properties needed are

$$LatchSet(i+1) \equiv$$

$$\Box \neg ResetInput(i+1) \ \& \ \nabla SetInput(i+1) \supset \nabla \Box SetState(i+1)$$

and

$$LatchReset(i) \equiv \Box \neg SetInput(i) \ \& \ \nabla ResetInput(i) \supset \nabla \Box ResetState(i).$$

(1) $\vdash A_1 \supset P_{i+1} \ \& \ Q_i$

(2) $\vdash P_{i+1} \ \& \ Q_i \twoheadrightarrow \neg \bar{S}_{i+1}$ by axiom G1

(3) $\vdash P_{i+1} \ \& \ Q_i \supset \nabla \neg \bar{S}_{i+1}$ by $\Box w \supset w$

(4) $\vdash A_1 \supset \nabla \neg \bar{S}_{i+1}$ by $\supset$ Trans. using (1),(3)

(5) $\vdash A_1 \supset \Box \neg ResetInput(i+1)$

(6) $\vdash A_1 \supset \Box \neg ResetInput(i+1) \ \& \ \nabla \neg \bar{S}_{i+1}$ by the PC tautology

$(w \supset u) \,\&\, (w \supset v) \equiv w \supset (u \,\&\, v)$

using (5),(4)

(7) $\vdash A_1 \supset \nabla \Box SetState(i+1)$ by $\supset$ Trans. using (6),$LatchSet(i+1)$

Line 7 says that, with the initial assumptions A_1, latch $i+1$ will eventually reach the set state and stay there forever. Line 7 is a subgoal needed to achieve another subgoal which concerns latch i.

$$\vdash A_1 \supset \nabla \Box ResetState(i).$$

(8) $\vdash A_1 \supset (\nabla \Box Q_{i+1} \,\&\, \nabla \Box \neg \bar{Q}_{i+1})$ by T8 and def. of $SetState(i+1)$

using (7)

(9) $\vdash A_1 \supset \nabla \Box ResetInput(i)$ by PC tautology

$(w \supset (u \,\&\, v)) \supset (w \supset u)$

and $\bar{Q}_{i+1} \equiv \bar{R}_i \equiv \neg ResetInput(i)$

using (8)

(10) $\vdash \nabla \Box ResetInput(i) \supset$

$\nabla ResetInput(i)$ by $\nabla \Box w \supset w$

(A5 and ∇ ∇ Rule)

(11) $\vdash A_1 \supset \nabla ResetInput(i)$ by $\supset$ Trans. using (9),(10)

(12) $\vdash A_1 \supset \Box \neg SetInput(i)$

(13) $\vdash A_1 \supset \Box \neg SetInput(i) \,\&$

$\nabla ResetInput(i)$ by PC tautology

$(w \supset u) \,\&\, (w \supset v) \equiv w \supset (u \,\&\, v)$

using (12),(11)

(14) $\vdash A_1 \supset \nabla \Box ResetState(i)$ by $\supset$ Trans. using (13),$LatchReset(i)$

The low $\bar{Q}_{i+1}$ output of latch $i+1$ resets latch i and this is modelled by the dependency of the goal of line 14 on the goal of line 7 in lines 8 through 14.

To get from the global machine assumption A_0 to the assumption about the dynamic portion A_1 requires the use of one inference rule and one static latch property. The part of A_1 which is not implied directly by A_0 is

$$\Box \neg SetInput(i) \ \& \ \Box \neg ResetInput(i+1). \tag{4.16}$$

This formula consists of two assertions of *non-interference*. They state that the dynamic portion is not affected by any signals in the rest of the machine. These non-interference properties are analogous to non-interference properties in verification of concurrent programs with shared variables [Owi76]. They are proven as follows.

(15) $\vdash A_0 \supset \Box \neg P_i$

(16) $\vdash \Box \neg P_i \supset \Box \neg P_i \vee \Box \neg Q_{i-1}$ by the PC tautology $u \supset u \vee v$

(17) $\vdash \Box \neg P_i \vee \Box \neg Q_{i-1} \supset$
$\Box (\neg P_i \vee \neg Q_{i-1})$ by T0

(18) $\vdash A_0 \supset \Box (\neg P_i \vee \neg Q_{i-1})$ by $\supset$ Trans. twice using (15),(16),(17)

(19) $\vdash \bar{S}_i \Rightarrow \bar{S}_i \ U_A \ (P_i \ \& \ Q_{i-1})$ by G3

(20) $\vdash A_0 \supset [\bar{S}_i \Rightarrow \bar{S}_i \ U_A \ (P_i \ \& \ Q_{i-1})]$ by the PC tautology $w \supset (v \supset w)$

(21) $\vdash A_0 \supset [\bar{S}_i \Rightarrow \Box \bar{S}_i]$ by SI using (18),(20)

Now the assumption *SetLinesSettled* is used on the entailment $\bar{S}_i \Rightarrow \Box \bar{S}_i$.

(22) $\vdash \Box(\bar{S}_i \supset \Box\bar{S}_i) \supset (\bar{S}_i \supset \Box\bar{S}_i)$ by $\Box w \supset w$

(23) $\vdash A_0 \supset (\bar{S}_i \supset \Box\bar{S}_i)$ by $\supset$ Trans. using (21),(22)

(24) $\vdash \bar{S}_i \supset (A_0 \supset \Box\bar{S}_i)$ by the PC tautology

$u \supset (v \supset w) \equiv v \supset (u \supset w)$

(25) $\vdash A_0 \supset \bar{S}_i$ by *SetLinesSettled*

(26) $\vdash A_0 \supset \Box\bar{S}_i$ by $\supset$ Trans. using (25),(24)

Since $\bar{S}_i$ is $\neg$ *SetInput*(i), one of the two non-interference assertions (formula (4.16)) has been proved. To prove the other assertion, steps 15 through 26 are repeated with $i+2$ substituted for i, yielding the following.

(27) $\vdash A_0 \supset \Box\neg SetInput(i+2)$

Further deductions on the results of lines 26 and 27 lead to the other goal.

(28) $\vdash A_0 \supset ResetState(i+2)$

(29) $\vdash A_0 \supset ResetState(i+2)$ &

$\Box\neg SetInput(i+2)$ by the PC tautology

$(w \supset u)$ & $(w \supset v) \equiv$

$w \supset (u$ & $v)$

using (27),(28)

(30) $\vdash ResetState(i+2)$ & $\Box\neg SetInput(i+2) \supset$

$\Box ResetState(i+2)$ by *LatchSet*

(31) $\vdash A_0 \supset \Box ResetState(i+2)$ by $\supset$ Trans. using (29),(30)

(32) $\vdash A_0 \supset \Box(\neg Q_{i+2}$ & $\bar{Q}_{i+2})$ by defn. of *ResetState*

(33) $\vdash A_0 \supset \Box\neg Q_{i+2}$ & $\Box\bar{Q}_{i+2}$ by T4

(34) $\vdash A_0 \supset \Box\neg ResetInput(i+1)$ by the equivalence $Q_{i+2} \equiv \overline{R}_{i+1} \equiv \neg ResetInput(i+1)$

This establishes the other non-interference assertion of formula (4.16).

Having now established that

(35) $\vdash A_0 \supset A_1$,

and that

(14) $\vdash A_1 \supset \nabla \Box ResetState(i)$

and

(7) $\vdash A_1 \supset \nabla \Box SetState(i+1)$,

it known that

$\vdash A_1 \supset \nabla \Box ResetState(i) \mathbin{\&} \nabla \Box SetState(i+1)$

by the PC tautology $(w \supset u) \mathbin{\&} (w \supset v) \equiv w \supset (u \mathbin{\&} v)$, and that

$\vdash A_0 \supset \nabla \Box ResetState(i) \mathbin{\&} \nabla \Box SetState(i+1)$.

Using the deduction schema of steps 15 through 26 again with the values 0, ..., $i-1$ and $i+2, \cdots, n-1$ substituted for i yields the $n-2$ implications

$$\vdash A_0 \supset \Box\neg SetInput(0)$$

$$\cdots$$

$$\vdash A_0 \supset \Box \neg SetInput(i-1)$$

$$\vdash A_0 \supset \Box \neg SetInput(i+2)$$

$$\cdots$$

$$\vdash A_0 \supset \Box \neg SetInput(n-1).$$

In addition, the following implications hold.

$$\vdash A_0 \supset ResetState(0)$$

$$\cdots$$

$$\vdash A_0 \supset ResetState(i-1)$$

$$\vdash A_0 \supset ResetState(i+2)$$

$$\cdots$$

$$\vdash A_0 \supset ResetState(n-1)$$

Using the PC tautology $(w \supset u)\ \&\ (w \supset v) \equiv w \supset (u\ \&\ v)$ the implications are combined.

$$\vdash A_0 \supset ResetState(0)\ \&\ \Box \neg SetInput(0)$$

$$\cdots$$

$$\vdash A_0 \supset ResetState(i-1)\ \&\ \Box \neg SetInput(i-1)$$

$$\vdash A_0 \supset ResetState(i+2)\ \&\ \Box \neg SetInput(i+2)$$

. . .

$$\vdash A_0 \supset \mathit{ResetState}(n-1) \ \& \ \Box \neg \mathit{SetInput}(n-1)$$

Combining lines 14, 7, and the above $n-2$ implications yields

$$\vdash A_0 \supset \Box \mathit{ResetState}(0) \ \& \ \cdots \ \& \ \Box \mathit{ResetState}(i-1)$$

$$\nabla \ \Box \mathit{ResetState}(i) \ \& \ \nabla \ \Box \mathit{SetState}(i+1)$$

$$\Box \mathit{ResetState}(i+2) \ \& \ \cdots \ \& \ \Box \mathit{ResetState}(n-1).$$

The final step is $n-2$ applications of theorem T34, yielding

$$\vdash A_0 \supset \nabla \ \Box (\mathit{ResetState}(0) \ \& \ \cdots \ \& \ \mathit{ResetState}(i-1)$$

$$\& \ \mathit{ResetState}(i) \ \& \ \mathit{SetState}(i+1)$$

$$\& \ \mathit{ResetState}(i+2) \ \& \ \cdots \ \& \ \mathit{ResetState}(n-1)).$$

Since it is now known that

$$\vdash \mathit{InState}(q_i) \ \& \ \mathit{Input}(p_{i+1}) \ \& \ \mathit{SetLinesSettled} \supset \nabla \ \Box \mathit{InState}(q_{i+1}),$$

it can be generalized using $\Box$ I. The eventuality is obviously true for any initial state $q_i \in Q$. So this completes the proof of the *StateChange* assertion.

CHAPTER 5

Global-Time Properties

In this chapter the focus turns to properties of asynchronous circuits which are of interest throughout the entire time of *normal* operation: global-time properties. In contrast to steady-state properties where stable eventualities take on the form

$$\Box \mathit{Stimulus} \supset \nabla \Box \mathit{Response},$$

and time is viewed locally, the global-time assumption dictates that stable eventualities (as in " $\nabla \Box \mathit{Response}$") are not stable forever, otherwise the circuit is in some kind of deadlocked situation where further stimulus will not produce corresponding responses. With steady-state analysis the stimulus is applied forever and the expected response is infinite stability. In contrast, the term *global-time* is used to denote when the time of application of the stimulus and the time of stability of the response are limited. Liveness properties then take the form

$$IC \ \& \ (\mathit{Stimulus}\ U_\Lambda\ \mathit{LongEnough}) \supset \nabla (\mathit{Response}\ U_\Lambda\ \mathit{NextStimulus})$$

where IC is a formula describing the initial condition (initial state), $\mathit{LongEnough}$ is a formula which is true when the stimulus has been sufficiently recognized by the

circuit, and *NextStimulus* is true when nothing more can be assumed regarding the stability of the circuit's response. For a latch, *LongEnough* would be a formula which is true when the latch has completed its response to a set or reset request (i.e. when Q and $\bar{Q}$ are at appropriate values). *NextStimulus* would describe the complimentary input situation of *Stimulus*, i.e. if the stimulus is a reset request then *NextStimulus* describes a set request. For our familiar $\bar{S}$–$\bar{R}$ latch of Figure 4.7, this would be expressed

$$(\neg \bar{R} \ \& \ \bar{S} \ U_A \ \neg Q \ \& \ \bar{Q}) \supset \nabla(\neg Q \ \& \ \bar{Q} \ U_A \ \neg \bar{S}).$$

For the asynchronous controller the liveness would be expressed

$$StateChange \equiv_{def} InState(q_i) \ \& \ (Input(p_{i+1}) \ U_A \ (InState(q_{i+1}) \ \& \ InputBeh)) \supset$$

$$\nabla \ (InState(q_{i+1}) \ U_A \ Input(q_{i+2})).$$

where "$InState(q_i)$" is the initial condition which must be assumed in order for the consequent to be true.

5.1 Proof Tools

5.1.1 Forward and Backward Reasoning

Let a *situation* be a PC formula that partially describes the state of a circuit.

When proving properties about the sequences of events in asynchronous circuits, two very important event relationships which must be expressible are:

(i) when situation r_1 is present, it causes situation r_2 to occur (liveness) and

(ii) r_2 will not occur until r_1 has occurred (safety).

An example of relationship (i) is expressed at the gate level by axioms G1 and G2. Relationship (ii) is expressed by axioms G3 and G4.

Forward reasoning

Forward Reasoning provides a way to derive the causal relationship of situations when composing devices. Forward reasoning is based on the following idea. Let formulas of the form r_i be true when a certain situation is true about a line of a circuit. If a situation r_1 implies the eventual occurrence of situation r_2, and situation r_2 implies the eventual occurrence of r_3, then it is true that situation r_1 implies the eventual occurrence of situation r_3. In a temporal logic framework, this corresponds to the transitivity property of eventualities of the form $u \twoheadrightarrow v$. The forward reasoning rule FR is stated

$$\vdash (w_1 \twoheadrightarrow w_2) \;\&\; (w_2 \twoheadrightarrow w_3) \supset (w_1 \twoheadrightarrow w_3). \tag{FR}$$

It is justified by the transitivity of the operator combination $\supset \nabla$ and the yields operator $\twoheadrightarrow$, established as follows. (We use transitivity of implication so much that we often do not mention that it was used or we abbreviate it simply as "trans.")

$\vdash \; w_1 \supset \nabla w_2$

$\vdash \; w_2 \supset \nabla w_3$

$\vdash \; \nabla w_2 \supset \nabla \nabla w_3$ by $\nabla\ \nabla$ Rule

$\vdash \; \nabla w_2 \supset \nabla w_3$ by T2

$\vdash \; w_1 \supset \nabla w_3$ by trans.

This establishes transitivity of $\supset \nabla$. The transitivity of $\supset \nabla$ is used to show the

transitivity of $\twoheadrightarrow$.

$\vdash\ w_1 \twoheadrightarrow w_2$

$\vdash\ w_1 \supset \nabla w_2$ by $\Box$E

$\vdash\ w_2 \twoheadrightarrow w_3$

$\vdash\ w_2 \supset \nabla w_3$ by $\Box$E

$\vdash\ w_1 \supset \nabla w_3$ by trans.

$\vdash\ w_1 \twoheadrightarrow w_3$ by $\Box$I

Backward reasoning

Backward reasoning provides a way to derive the safety relationship between events when composing devices. The safety relationship between two related situations dictates that if situation r_1 precedes situation r_2 then r_2 cannot occur until r_1 occurs. Backward reasoning is the property that if r_1 precedes r_2 and r_2 precedes r_3 then r_1 precedes r_3. The U_A operator is used to express this safety as in $(\neg r_2\, U_A\, r_1)$. Backward reasoning (BR) is inherent in an U_A-property called *restricted transitivity.* One way to state restricted transitivity is

$$\vdash (w_1\, U_A\, w_2)\ \&\ (\neg w_2\, U_A\, w_3) \supset (w_1\, U_A\, w_3). \qquad \text{(RT)}$$

Restated in terms of situations, it becomes

$$\vdash (\neg r_2\, U_A\, r_1)\ \&\ (\neg r_3\, U_A\, r_2) \supset (\neg r_3\, U_A\, r_1) \qquad \text{(RTS)}$$

Instead of RT above, a more general and useful form will be used from now on. The restricted transitivity of U_A, now stated more generally as

$$\vdash (w_1\, U_A\, w_2)\ \&\ (w_3\, U_A\, w_4)\ \&\ \Box\neg(w_2\ \&\ w_3) \supset (w_1\, U_A\, w_4), \quad \text{(BR)}$$

is established by reasoning that if w_2 occurs at time i and w_4 occurs at time j then, since w_2 and w_3 are in conflict (they can never be true at the same point in time as dictated by $\Box\neg(w_2\ \&\ w_3)$), $i>j$ and so w_1 is true until w_4. This situation is pictured in Figure 5.1. The term "restricted" refers to the fact that the transitivity is valid only when w_2 and w_3 are in conflict. U_A is not transitive in the ordinary sense.

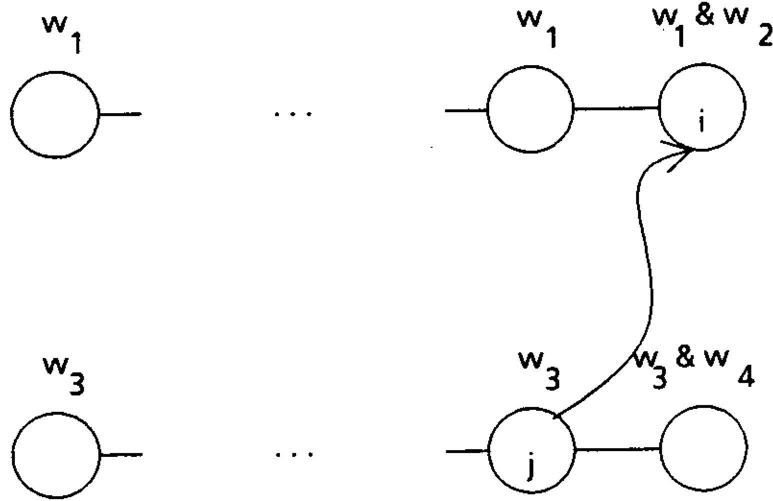


Figure 5.1 A Valid Model for Restricted U_A Transitivity
The arrow from the node labelled "j" to the node labelled "i" indicates that j must precede i.

BR can be proved formally by using a Wolper-style (like A9) axiom for U_A :

$$\vdash u\ \&\ \Box(u \supset ((p\ \&\ q) \vee (p\ \&\ \bigcirc u))) \supset p\, U_A\, q$$

with $u \equiv (w_1\, U_A\, w_2)\ \&\ (w_3\, U_A\, w_4)\ \&\ \Box\neg(w_2\ \&\ w_3)$ as the induction

hypothesis.

BR can be used to establish RTS by substituting

$\neg r_2$ for w_1,

r_1 for w_2,

$\neg r_3$ for w_3,

r_2 for w_4.

BR will be the form used from now on.

Generalizing the BR rule using the restricted transitivity property of U_A yields the generalized backward reasoning (GBR) rule stated

$$\vdash \bigwedge_{i=1}^{k-1} \Box \neg (u_i \ \& \ v_{i+1}) \ \& \ \bigwedge_{i=1}^{k} (u_i \ U_A \ v_i) \supset (\bigwedge_{i=1}^{k} u_i) \ U_A \ v_1 \qquad \text{(GBR)}$$

and justified by BR used $k-1$ times.

Those familiar with program correctness will notice that forward reasoning is somewhat analogous to the forward assignment axiom of Floyd [Flo67], and backward reasoning is analogous to the backward assignment axiom of Hoare [Hoa69]. To see this, picture a chain of inverters as analogous to the sequence of assignments

$$x_2 := x_1 \ ; \ \cdots \ ; \ x_n := x_{n-1}$$

When composing individual devices to form a circuit, we need to obtain new specifications for the composite circuit and prove that it will behave according to the new specifications. Here some small example circuits are considered, and a taste of the usage of FR and BR is presented. The example of a delay element consisting of an inverter chain is used to illustrate forward and backward reasoning.

An inverter with input line L and output line $\bar{L}$ would have the following gate axioms.

$$\vdash L \twoheadrightarrow \neg \bar{L} \tag{G1}$$

$$\vdash \neg L \twoheadrightarrow \bar{L} \tag{G2}$$

$$\vdash \bar{L} \Rightarrow \bar{L}\ U_A\ L \tag{G3}$$

$$\vdash \neg \bar{L} \Rightarrow \neg \bar{L}\ U_A\ \neg L \tag{G4}$$

$$\vdash L \Rightarrow L\ U_A\ \neg \bar{L} \tag{G5}$$

$$\vdash \neg L \Rightarrow \neg L\ U_A\ \bar{L} \tag{G6}$$

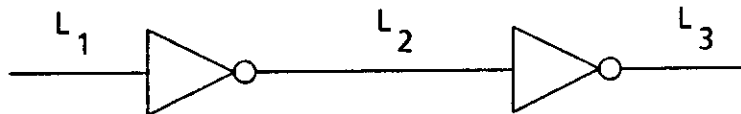


Figure 5.2 A Two-Inverter Delay Element

Consider a simple delay element consisting of the composition of two inverters, shown in Figure 5.2. The liveness property pair

$$L_1 \twoheadrightarrow L_3 \qquad \neg L_1 \twoheadrightarrow \neg L_3$$

can be proved directly by forward reasoning (the transitivity of $\twoheadrightarrow$). The safety

property pair

$$\neg L_2 \mathbin{\&} L_3 \Rightarrow L_3\, U_A\, \neg L_1 \qquad L_2 \mathbin{\&} \neg L_3 \Rightarrow L_3\, U_A\, L_1$$

can be proved as follows. Let $Settled_1 \equiv_{def} \neg L_2 \mathbin{\&} L_3$.

$$\vdash \neg L_2 \supset \neg L_2\, U_A\, \neg L_1$$

$$\vdash L_3 \supset L_3\, U_A\, L_2$$

$$\vdash Settled_1 \supset \neg L_2\, U_A\, \neg L_1$$

$$\vdash Settled_1 \supset L_3\, U_A\, L_2$$

$$\vdash Settled_1 \supset L_3\, U_A\, \neg L_1$$

$$\vdash \neg L_2 \mathbin{\&} L_3 \Rightarrow L_3\, U_A\, \neg L_1$$

The second-to-last step is justified by backward reasoning. The dual property follows similarly. Note that in the final formula the assumption $\neg L_2 \mathbin{\&} L_3$ is made. This is required to guarantee that the neither of the gates are initially in the excited state (where the output has not yet reacted to an input change). The slightly more elegant formula

$$\vdash L_3 \Rightarrow L_3\, U_A\, \neg L_1$$

would not be derivable and rightfully so. It would allow L_2 to be high, and this would mean that the second inverter is excited. If the second inverter is excited then the safety property is not valid since L_3 could go low before L_1 goes low due to the unexpected 1 on line L_2. Internal line L_2 must be have a value that is the compliment of line L_3 for the safety property to be valid.

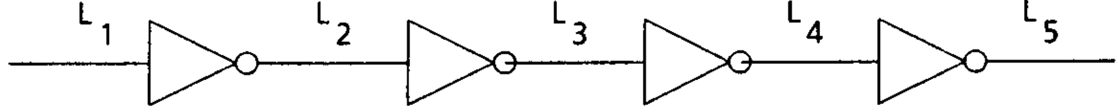


Figure 5.3 A Four-Inverter Delay Element

The same two-inverter delay element can be composed with itself again, producing a longer delay (and possibly an amplification of the signal). The delay element in Figure 5.3 is composed of four inverters. Again the liveness property pair

$$L_1 \twoheadrightarrow L_5 \qquad \neg L_1 \twoheadrightarrow \neg L_5$$

can be proved directly from the transitivity of $\twoheadrightarrow$. The safety property pair

$$\neg L_2 \,\&\, L_3 \,\&\, \neg L_4 \,\&\, L_5 \;\Rightarrow\; L_5\, U_A\, \neg L_1$$

$$L_2 \,\&\, \neg L_3 \,\&\, L_4 \,\&\, \neg L_5 \;\Rightarrow\; \neg L_5\, U_A\, L_1$$

can be proved as follows. Let $Settled_0' \equiv_{def} \neg L_2 \,\&\, L_3 \,\&\, \neg L_4 \,\&\, L_5$. Then

$$\vdash\; L_2 \,\&\, \neg L_3 \supset \neg L_3\, U_A\, L_1$$

$$\vdash\; L_4 \,\&\, \neg L_5 \supset \neg L_5\, U_A\, L_3$$

$$\vdash\; Settled_0' \supset \neg L_3\, U_A\, L_1$$

$$\vdash\; Settled_0' \supset \neg L_5\, U_A\, L_3$$

$$\vdash\; Settled_0' \supset \neg L_5\, U_A\, L_1$$

$$\vdash \neg L_2 \,\&\, L_3 \,\&\, \neg L_4 \,\&\, L_5 \Rightarrow L_5 \, U_A \, \neg L_1.$$

The second-to-last step is justified by BR.

GBR could also be used to derive the inverter chain safety properties in just 1 step. We go from

$$\vdash Settled_0' \supset \neg L_2 \, U_A \, \neg L_1$$

$$\& \, L_3 \, U_A \, L_2$$

$$\& \, \neg L_4 \, U_A \, \neg L_3$$

$$\& \, L_5 \, U_A \, L_4$$

to

$$\vdash Settled_0' \supset Settled_0' \, U_A \, \neg L_1.$$

The latter formula states that, initially in $Settled_0'$, the chain will remain in that state until disrupted by L_1 going low.

So far inverter chains have been the only circuits considered. Composing the various logic gates poses a problem called *stage synchronization,* which is an artifact of the unbounded delay assumption. Consider, for example, the two AND gate chain of Figure 5.3. If it is known that X, Y, and Z are true now, it often desirable to conclude that V will be true some time later. If Z goes low during the time that the first AND gate is switching, however, T will be high and Z low for the second AND gate, causing V to become (or remain) low. Since the Z line bypasses the first gate, another condition must be added to sychronize Z with T. If it is known that $(Z \; U_A \; T)$ then by the coincidence rule (CR) we may conclude that $\nabla (Z \,\&\, T)$. Then from the

liveness gate axiom for the second AND gate, $Z \,\&\, T \twoheadrightarrow V$, the property $\nabla (Z \,\&\, T) \supset \nabla V$ can be obtained by elimination of the $\Box$ (T1) and application of the ∇ ∇ Rule. Then by transitivity of implication

$$\vdash X \,\&\, Y \,\&\, (Z\ U_A\ T) \supset \nabla V$$

follows.

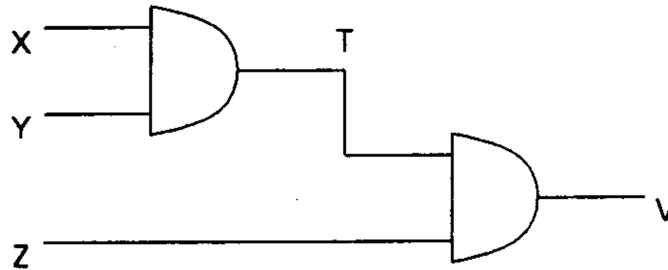


Figure 5.4 Combinational Logic the with Stage Sychronization Problem

5.1.2 Transition Safety Properties

As a tool in proving certain liveness properties when the stage sychronization problem (Section 5.1.1) is present, safety properties of the form

$$\neg \beta\ U^E\ (\beta\ U_A \neg \alpha) \tag{5.1}$$

are used. Formula (5.1) is a stronger substitute for the weaker

$$\nabla (\beta\ U_A \neg \alpha) \tag{5.2}$$

which often appears in safety proofs. Formula (5.1) not only says eventually β is true until $\neg \alpha$, but also that β is not true up until that time.

A formula like (5.1) is called a *Transition Safety Property (TSP)*. TSPs are safety properties of propositions undergoing a transition. A TSP like (5.1) is a statement about the safety of proposition β (which could be a line name or a propositional logic formula) during a transition from $\neg\beta$ to β. We call such a TSP a 1-TSP because of the single transition. An n-TSP is a property about a proposition undergoing n transitions.

Using this definition of n-TSP then, we would conclude that the gate safety axioms G3 and G4 are 0-TSPs. We could formulate 1-TSPs like

$$\neg\beta \Rightarrow \neg\beta\ U\ \beta\ U_A\ \neg\alpha.$$

and 2-TSPs like

$$\beta \Rightarrow \beta\ U\ \neg\beta\ U\ \beta\ U_A\ \neg\alpha.$$

and so on for gates. Note that each U operator may or may not be strong, but the U_A is usually weak, allowing for the proposition β to be in its final state forever.

5.1.3 Proof Rules

Global-time versions of the proof rules of Chapter 4 are similar their steady-state counterparts, but with the steady-state stability formulas of the form $\Box u$ replaced with the global-time version $(u\ U_A\ \gamma)$ where γ is true when the stability was in effect long enough. The rules concern finite stability, and therefore we prepend the word "finite" to the names of their steady-state counterparts to arrive at the names of the global-times rules.

Finite Inverter-Gate Stability:

$$\vdash w \supset \beta_1 \,\&\, \beta_2 \tag{FIGS}$$

$$\vdash w \supset \beta_1 \Rightarrow \beta_1\, U_A\, (\neg \beta_2 \vee \gamma)$$

$$\frac{\vdash w \supset \beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1)}{\vdash w \supset (\beta_1 \,\&\, \beta_2)\, U_A\, \gamma}$$

Finite Gate-Gate Stability:

$$\vdash w \supset \beta_1 \,\&\, \beta_2 \tag{FGGS}$$

$$\vdash w \supset \beta_1 \Rightarrow \beta_1\, U_A\, (\neg \beta_2 \vee \gamma)$$

$$\frac{\vdash w \supset \beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1 \,\&\, \alpha)}{\vdash w \supset (\beta_1 \,\&\, \beta_2)\, U_A\, \gamma}$$

Finite Eventual Gate-Gate Stability

$$\vdash w \supset \nabla (\beta_1 \,\&\, \beta_2) \tag{FEGGS}$$

$$\vdash w \supset \beta_1 \Rightarrow \beta_1\, U_A\, (\neg \beta_2 \vee \gamma)$$

$$\frac{\vdash w \supset \beta_2 \Rightarrow \beta_2\, U_A\, (\neg \beta_1 \,\&\, \alpha)}{\vdash w \supset \nabla ((\beta_1 \,\&\, \beta_2)\, U_A\, \gamma)}$$

5.2 Safety Strategy

The strategy for proving global-time safety is simply to use BR and GBR, aided by any PTL theorems needed. This is illustrated in the safety of a latch and the safety of the asynchronous controller.

5.2.1 Latch Safety

With the global-time assumption in mind, the safety properties of latches can be established. The latch safety properties

$$LatchStayReset \equiv_{def} Reset \Rightarrow Reset\ U_A \neg \bar{S}$$

$$LatchStaySet \equiv_{def} Set \Rightarrow Set\ U_A \neg \bar{R}$$

where $Reset \equiv_{def} \neg Q \;\&\; \bar{Q}$ and $Set \equiv_{def} Q \;\&\; \neg \bar{Q}$ state that nothing will occur to the outputs of the latch until an appropriate input is pulled low. *LatchStayReset* says that if the latch is reset now then it will remain so until a new stimulus from $\bar{S}$ occurs. *LatchStayReset* is easily proved using FGGS. Assuming $A_0 \equiv_{def} Reset$ we have

$\vdash A_0 \supset \neg Q \;\&\; \bar{Q}$

$\vdash A_0 \supset \neg Q \Rightarrow \neg Q\ U_A\ (\neg \bar{S} \vee \neg \bar{Q})$

$\vdash A_0 \supset \bar{Q} \Rightarrow \bar{Q}\ U_A\ (Q \;\&\; \bar{R})$

$\vdash A_0 \supset Reset\ U_A \neg \bar{S}$ by FGGS

5.2.2 Controller Safety

Figure 5.4 shows one cell of the controller, $Cell_k$. For the controller, safety is the property of staying in state q_{i+1} after the state transition $q_i \rightarrow q_{i+1}$ until the next input, p_{i+2}, occurs.

$$\vdash InputBeh \;\&\; IS(q_{i+1}) \Rightarrow IS(q_{i+1})\ U_A\ I(p_{i+2}) \tag{5.3}$$

where we abbreviate *InState* as *IS* and *Input* as *I*. In proving (5.3), the *InputBeh*

assumption is necessary to guarantee that (the notation of Chapter 4 is used here)

$$\neg P_{i+2} \, U \, I(p_{i+2}). \tag{5.4}$$

It is not guarateed that the controller will be stable in new state q_{i+1} unless P_{i+2} stays low. *InputBeh* is an assumption that guaratees that after an input is applied, there is a period when no input is applied, i.e. a period when *NoInput* defined

$$NoInput \equiv_{def} \bigwedge_{i=1}^{n} \neg P_i$$

is applied. *InputBeh* is defined

$$I(p_{i+1}) \, U \, NoInput \, U \, I(p_{i+2})$$

Derivation of our goal (5.4) from *InputBeh* proceeds as follows.

$A_0 \equiv_{def} I(p_{i+1}) \, U \, NoInput \, U \, I(p_{i+2})$

(1) $\vdash \Box (I(p_{i+1}) \supset \neg P_{i+2})$ — by def. of I and $\Box$I

(2) $\vdash A_0 \supset \neg P_{i+2} \, U \, NoInput \, U \, I(p_{i+2})$ — by T25

$u_0 \equiv_{def} \neg P_{i+2} \, U \, NoInput \, U \, I(p_{i+2})$

(3) $\vdash u_0 \supset (NoInput \, U \, I(p_{i+2})) \vee (\neg P_{i+2} \,\&\, \bigcirc u_0)$ — by def. of U A8

(4) $\vdash \Box (NoInput \supset \neg P_{i+2})$ — by def. of *NoInput* and $\Box$I

(5) $\vdash u_0 \supset (\neg P_{i+2} \, U \, I(p_{i+2})) \vee (\neg P_{i+2} \,\&\, \bigcirc u_0)$ — by T25, PC using 3,4

(6) $\vdash \Box (u_0 \supset (\neg P_{i+2} \, U \, I(p_{i+2})) \vee (\neg P_{i+2} \,\&\, \bigcirc u_0))$ — by $\Box$I

$(7) \vdash A_0 \supset u_0 \;\&\; \Box(u_0 \supset (\neg P_{i+2}\, U\, I(p_{i+2}))$

$\vee (\neg P_{i+2} \;\&\; \bigcirc u_0))$ by PC using 2,6

$(8) \vdash A_0 \supset \neg P_{i+2}\, U\, \neg P_{i+2}\, U\, I(p_{i+2})$ by Wolper's A10

$(9) \vdash A_0 \supset \neg P_{i+2}\, U\, I(p_{i+2})$ by T27

Figure 5.5 shows the basic controller cell, $Cell_k$.

We now return to the derivation of (5.3). To the $n-2$ pairs of safety formulas

$$\vdash IS(q_{i+1}) \supset \neg Q_k\, U_A\, \neg \bar{S}_k$$

$$\vdash IS(q_{i+1}) \supset \bar{S}_k\, U_A\, (Q_{k-1} \;\&\; P_k),$$

derived from gate axioms and latch properties with $k = i, i-1, ..., i+2$, we apply the general backward reasoning rule GBR, yielding

$$\vdash IS(q_{i+1}) \supset (\neg Q_i \;\&\; \neg Q_{i-1} \;\&\; \cdots \;\&\; \neg Q_{i+2})\, U_A\, (Q_{i+1} \;\&\; P_{i+2})$$

and

$$\vdash IS(q_{i+1}) \supset (\bar{S}_i \;\&\; \bar{S}_{i-1} \;\&\; \cdots \;\&\; \bar{S}_{i+2})\, U_A\, (Q_{i+1} \;\&\; P_{i+2}).$$

These, combined with formula (5.4) using BR yields

$$\vdash IS(q_{i+1}) \supset (\neg Q_i \;\&\; \neg Q_{i-1} \;\&\; \cdots \;\&\; \neg Q_{i+2})\, U_A\, I(p_{i+2}) \qquad (5.5)$$

and

$$\vdash IS(q_{i+1}) \supset (\bar{S}_i \;\&\; \bar{S}_{i-1} \;\&\; \cdots \;\&\; \bar{S}_{i+2})\, U_A\, I(p_{i+2}), \qquad (5.6)$$

leaving Q_{i+1} and $\bar{S}_{i+1}$ yet to be included in these two safety formulas.

Note that the $Cell_{i+1}$ $\bar{S}$ line, $\bar{S}_{i+1}$, is left out of these two formulas. This is because $Cell_{i+1}$ is in a different state from the others. It will be treated shortly.

For each $k = i, i-1, ..., i+2$, we apply BR and T28, taking the $n-2$ pairs

$$\vdash IS(q_{i+1}) \supset \bar{Q}_j \; U_A \neg \bar{S}_j$$

$$\vdash IS(q_{i+1}) \supset \bar{S}_j \; U_A \; I(p_{i+2})$$

into

$$\vdash IS(q_{i+1}) \supset (\bar{Q}_i \;\&\; \bar{Q}_{i-1} \;\&\; \cdots \;\&\; \bar{Q}_{i+2}) \; U_A \; I(p_{i+2}). \tag{5.7}$$

Again the $Cell_{i+1}$ line is left out, $\bar{Q}_{i+1}$

Now we work on including the three line variables Q_{i+1}, $\bar{S}_{i+1}$, and $\bar{Q}_{i+1}$ in the safety formulas. >From formula (5.7) we know

$$\vdash IS(q_{i+1}) \supset \bar{Q}_{i+2} \; U_A \; I(p_{i+2}).$$

Using the *LatchStaySet* safety property, we also know

$$\vdash IS(q_{i+1}) \supset Q_{i+1} \; U_A \neg \bar{Q}_{i+2}.$$

We use BR again on these two, then include the result into formula (5.5) with T28, yielding

$$\vdash IS(q_{i+1}) \supset (\neg Q_i \;\&\; \neg Q_{i-1} \;\&\; \cdots \;\&\; \neg Q_{i+2} \;\&\; Q_{i+1}) \; U_A \; I(p_{i+2}) \tag{5.8}$$

Using BR on the pairs

$$\vdash IS(q_{i+1}) \supset \bar{S}_{i+1} \; U_A \; Q_i$$

$$\vdash IS(q_{i+1}) \supset \neg Q_i \; U_A \; I(p_{i+2})$$

and

$$\vdash IS(q_{i+1}) \supset \neg \overline{Q}_{i+1} \; U_A \; \neg \overline{Q}_{i+2} .$$

$$\vdash IS(q_{i+1}) \supset \overline{Q}_{i+2} \; U_A \; I(p_{i+2})$$

and using T28 to combine these results with (5.6), (5.7) and (5.8) yields (5.3), the desired safety formula for the controller.

5.3 Liveness Strategy

Proving liveness is much less straightforward than proving safety. A combination of forward and backward reasoning is used, and PTL timing diagrams are used to clarify the ordering of signal transitions. The heuristics used in proving the liveness of a circuit such as our controller are

Liveness Strategy	
1.	Determine the active and non-active portions of the circuit
2.	Partition the circuit into cells which are of manageable size
3.	Prove cell properties
4.	Prove non-interference properties
5.	Determine any 1-TSPs
6.	Use PTL timing diagrams for intuitive reasoning

5.3.1 Latch Liveness

Liveness of the latch is can be expressed in several ways. Adhering to the general form of global-time liveness, liveness should be stated

$$((\neg \bar{S} \ \& \ \bar{R})\ U_A\ Set) \twoheadrightarrow (Set\ U_A \neg \bar{R})$$

This form is provable in our system. A weaker form,

$$(\neg \bar{S} \ \& \ (\bar{R}\ U_A\ Set)) \twoheadrightarrow (Set\ U_A \neg \bar{R}),$$

is also provable. In practice, though, an even weaker form is the most useful.

$$LatchSet \equiv_{def} (\nabla \neg \bar{S} \ \& \ (\bar{R}\ U_A\ Set)) \twoheadrightarrow (Set\ U_A \neg \bar{R}),$$

This form allows the $\bar{S}$ stimulus to occur anytime from now into the future, assuming that $\bar{R}$ remains high during that time. This is useful when the $\bar{S}$ input is the output of a gate, as in the controller example.

In the derivation of *LatchSet*, let the initial assumptions be

$$A_0 \equiv_{def} \nabla \neg \bar{S} \ \& \ (\bar{R}\ U_A\ Set)$$

The proof now follows.

(1) $\vdash A_0 \supset \nabla \neg \bar{S}$	
(2) $\vdash \nabla \neg \bar{S} \supset \nabla (\neg \bar{S} \vee \neg \bar{Q})$	by PC and ∇ ∇ Rule
(3) $\vdash \neg \bar{S} \vee \neg \bar{Q} \twoheadrightarrow Q$	gate axiom
(4) $\vdash \nabla (\neg \bar{S} \vee \neg \bar{Q}) \supset \nabla Q$	∇ ∇ Rule
(5) $\vdash A_0 \supset \nabla Q$	trans. on 1,2,4
(6) $\vdash A_0 \supset (\bar{R}\ U_A\ Set)$	

$(7) \vdash (\bar{R}\ U_A\ Set) \supset \Box \bar{R}$

$\vee ((\bar{R}\ U\ (\bar{R} \ \&\ Q \ \&\ \neg \bar{Q}))$

$\&\ \nabla (\bar{R} \ \&\ Q \ \&\ \neg \bar{Q}))$ by def. of U_A

$(8) \vdash A_0 \supset (\nabla Q \ \&\ \Box \bar{R})$

$\vee (\nabla Q \ \&\ (\bar{R}\ U\ (\bar{R} \ \&\ Q \ \&\ \neg \bar{Q}))$

$\&\ \nabla (\bar{R} \ \&\ Q \ \&\ \neg \bar{Q}))$ by PC using 5,6,7

$(9) \vdash \nabla Q \ \&\ \Box \bar{R} \supset \nabla (Q \ \&\ \bar{R})$ by T7

$(10) \vdash \nabla Q \ \&\ (\bar{R}\ U\ (\bar{R} \ \&\ Q \ \&\ \neg \bar{Q}))$

$\&\ \nabla (\bar{R} \ \&\ Q \ \&\ \neg \bar{Q})) \supset$

$\nabla (Q \ \&\ \bar{R})$ by T6

$(11) \vdash A_0 \supset \nabla (Q \ \&\ \bar{R})$ by PC using 8,9,10

$(12) \vdash Q \ \&\ \bar{R} \Rightarrow Q \ \&\ \bar{R}\ U_A \neg \bar{Q}$ gate axiom

$(13) \vdash Q \ \&\ \bar{R} \twoheadrightarrow \neg \bar{Q}$ gate axiom

$(14) \vdash Q \ \&\ \bar{R} \supset ((Q \ \&\ \bar{R})\ U_A \neg \bar{Q}) \ \&\ \nabla \neg \bar{Q}$ by T4 and PC using 12,13

$(15) \vdash ((Q \ \&\ \bar{R})\ U_A \neg \bar{Q}) \ \&\ \nabla \neg \bar{Q}$

$\supset \nabla (Q \ \&\ \bar{R} \ \&\ \neg \bar{Q})$ by CRimp

$(16) \vdash Q \ \&\ \bar{R} \supset \nabla (Q \ \&\ \bar{R} \ \&\ \neg \bar{Q})$ by trans. using 14,15

$(17) \vdash Q \ \&\ \bar{R} \supset \nabla (Q \ \&\ \neg \bar{Q})$ by T6

$(18) \vdash \nabla (Q \ \&\ \bar{R}) \supset \nabla (Q \ \&\ \neg \bar{Q})$ by $\nabla\ \nabla$ Rule and T2

$(19) \vdash A_0 \supset \nabla (Q \ \& \ \neg \bar{Q})$ trans. using 11,18

$(20) \vdash A_0 \supset \neg \bar{Q} \Rightarrow \neg \bar{Q} \ U_A \ (\neg Q \vee \neg \bar{R})$ gate axiom

$(21) \vdash A_0 \supset Q \Rightarrow Q \ U_A \ (\bar{S} \ \& \ \bar{Q})$ gate axiom

$(22) \vdash A_0 \supset \nabla (Q \ \& \ \neg \bar{Q} \ U_A \ \neg \bar{R})$ by FEGGS using 19,20,21

Steps 1-11 are a derivation of the fact that applying the input will result in a future time when Q and $\bar{R}$ are both high. These being high will cause $\bar{Q}$ to go low (step 13). Input safety (step 12) guarantees that Q and $\bar{R}$ are high and $\bar{Q}$ is low at some point. This situation is stable by FEGGS (step 22).

The dual of *LatchSet*, *LatchReset*, is derived similarly.

5.3.2 Controller Liveness

For asynchronous controller, we have determined in Chapter 4 that the active portion of the circuit for the *StateChange* property consists of the two latches and two NAND gates labelled i and $i+1$. Figure 5.5 shows a single cell of the partitioning. The Cells have also been determined in the partitioning discussed in Chapter 4. The antecedent of *StateChange* is

$$A_{SC} \equiv_{def} InState(q_i) \ \& \ (Input(p_{i+1}) \ U_A \ (InState(q_{i+1}) \ \& \ InputBeh))$$

Next we turn to proving the cell properties.

The cell property of $Cell_i$ is simply *LatchReset*, as the latch is reset by $Cell_{i+1}$. The cell property of $Cell_{i+1}$ is proved by using the liveness of the NAND gate and *LatchSet*. Let us use

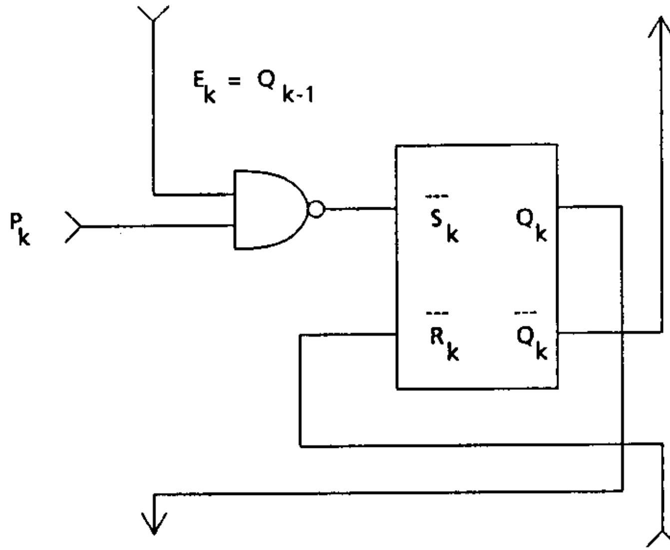


Figure 5.5. One Cell of the Controller

$$A_0 \equiv_{def} E_{i+1} \ \& \ P_{i+1} \ \& \ (\overline{R}_{i+1} \ U_A \ Set_{i+1})$$

as our initial assumptions. The proof of the cell property for $Cell_{i+1}$ is

$\vdash A_0 \supset E_{i+1} \ \& \ P_{i+1}$	by PC
$\vdash E_{i+1} \ \& \ P_{i+1} \supset \nabla \neg \overline{S}_{i+1}$	gate axiom and $\Box$ E
$\vdash A_0 \supset \nabla \neg \overline{S}_{i+1} \ \& \ (\overline{R}_{i+1} \ U_A \ Set_{i+1})$	by PC
$\vdash A_0 \supset \nabla \ (Set_{i+1} \ U_A \ \neg \overline{R}_{i+1})$	by *LatchSet*

Since $A_{SC} \supset A_0$, it is also known that

$$\vdash A_{SC} \supset \nabla \ (Set_{i+1} \ U_A \ \neg \overline{R}_{i+1}) \tag{5.9}$$

and

$$\vdash A_{SC} \supset \nabla \ (Reset_i \ U_A \ \neg \overline{S}_i) \tag{5.10}$$

from *LatchSet* and *LatchReset* respectively.

From the discussion of the steady-state behavior of the controller in Chapter 4, it was determined that $Cell_i$ and $Cell_{i+1}$ are the active cells during a state change. Interference from adjacent cells was ruled out by the non-interference properties. Non-interference properties are again established to allow focused reasoning about the active portion of the circuit. The two non-interference properties are then

$$\vdash \ A_{SC} \supset \bar{S}_i \ U_A \ IS(q_{i+1}) \tag{5.11}$$

$$\vdash \ A_{SC} \supset \bar{R}_{i+1} \ U_A \ IS(q_{i+1}) \tag{5.12}$$

(5.11) is proved as follows.

$\vdash A_{SC} \supset \bar{S}_i \ U_A \ (E_i \ \& \ P_i)$ gate axiom

$\vdash A_{SC} \supset \neg P_i \ U_A \ IS(q_{i+1})$ by def. of A_{SC}

$\vdash A_{SC} \supset \bar{S}_i \ U_A \ IS(q_{i+1})$ by BR

(5.12) is proved as follows.

(1) $\vdash A_{SC} \supset \neg Q_{i+2} \ \& \ \bar{Q}_{i+2} \ U_A \ \neg \bar{S}_{i+2}$ by *LatchStayReset*

(2) $\vdash A_{SC} \supset \bar{Q}_{i+2} \ U_A \ \neg \bar{S}_{i+2}$ by T28

(3) $\vdash A_{SC} \supset \bar{S}_{i+2} \ U_A \ (E_{i+2} \ \& \ P_{i+2})$ gate axiom

(4) $\vdash A_{SC} \supset \bar{Q}_{i+2} \ U_A \ (E_{i+2} \ \& \ P_{i+2})$ by BR using 2,3

(5) $\vdash A_{SC} \supset \neg P_{i+2} \ U \ IS(q_{i+1})$ by def. of A_{SC}

(6) $\vdash A_{SC} \supset \bar{Q}_{i+2} \ U \ IS(q_{i+1})$ by BR using 4,5

$(7) \vdash A_{SC} \supset \bar{R}_{i+1} \, U_A \, IS(q_{i+1})$ by $\bar{Q}_{i+2} \equiv \bar{R}_{i+1}$ and ER

Keeping with the liveness strategy, we next determine the appropriate 1-TSPs. We have found that it is a good idea to try to limit our ambition to proving only properties which involve 0 or 1 transition per line. *StateChange* does not quite meet this requirement. The $\bar{S}_{i+1}$ line changes from high to low, setting off the transitions in the active portion of the circuit, then is set back to high again. But if we define

$$AlmostInState(q_{i+1}) \equiv_{def} Q_{i+1} \,\&\, \neg \bar{Q}_{i+1} \,\&\, \left[\bigwedge_{j=i+2}^{l} (\neg Q_j \,\&\, \bar{Q}_j) \right] \,\&\, \left[\bigwedge_{j=i+2}^{i} \bar{S}_i \right]$$

(abbreviated $AIS(q_{i+1})$) and

$$InState(q_{i+1}) \equiv_{def} \bar{S}_{i+1} \,\&\, AlmostInState(q_{i+1}),$$

then prove the pair of liveness properties

$$\vdash IS(q_i) \,\&\, I' \supset \nabla (AIS(q_{i+1}) \,\&\, I') \tag{5.13}$$

$$\vdash AIS(q_{i+1}) \,\&\, I' \supset \nabla (IS(q_{i+1})) \tag{5.14}$$

where $I' \equiv_{def} (I(p_{i+1}) \, U_A \, (IS(q_{i+1}) \,\&\, InputBeh))$, we will find this more tractable.

The 1-TSP property of $\bar{S}_{i+1}$ is

$$A_{SC} \supset \bar{S}_{i+1} \, U \neg \bar{S}_{i+1} \, U_A \, (\neg E_{i+1} \vee \neg P_{i+1})$$

This can be assumed as a gate axiom. In fact, we simply augment the basic gate axioms with any 1-TSPs needed. The 1-TSPs for $Cell_{i+1}$ to be derived for $Cell_{i+1}$ and $Cell_i$ are

$$(Cell_{i+1}) \qquad A_{SC} \supset \neg Set_{i+1}\, U^E\, Set_{i+1}\, U_A\, IS(q_{i+1}) \qquad (5.15)$$

$$(Cell_i) \qquad A_{SC} \supset \neg Reset_i\, U^E\, Reset_i\, U\, IS(q_{i+1}). \qquad (5.16)$$

This proceeds as follows. (All line variables have their "$i+1$" subscripts omitted.)

(1) $\vdash A_{SC} \supset \bar{S}\, U \neg \bar{S}\, U_A\, (\neg E \vee \neg P)$	1-TSP gate axiom
(2) $\vdash A_{SC} \supset E\, U_A \neg \bar{Q}$	by *LatchStaySet*(i)
(3) $\vdash A_{SC} \supset P\, U_A \neg \bar{Q}$	by I'
(4) $\vdash A_{SC} \supset \bar{Q}\, U_A \neg \bar{S}$	by *LatchStayReset*$(i+1)$
(5) $\vdash A_{SC} \supset \bar{S}\, U \neg \bar{S}\, U_A \neg \bar{Q}$	by DP using 1,2,3,4
(6) $\vdash A_{SC} \supset \neg Q\, U\, Q\, U_A\, (\bar{S}\ \&\ \bar{Q})$	1-TSP gate axiom
(7) $\vdash A_{SC} \supset \bar{Q}\, U \neg \bar{Q}\, U_A\, (\neg Q \vee \neg \bar{R})$	1-TSP gate axiom
(8) $\vdash A_{SC} \supset \neg Q\, U_A\, (\neg \bar{S} \vee \neg \bar{Q})$	gate axiom (0-TSP)
(9) $\vdash A_{SC} \supset \bar{Q}\, U_A\, (Q\ \&\ \bar{R})$	gate axiom
(10) $\vdash A_{SC} \supset (\neg Q \vee \bar{Q})\, U$ $[(Q\, U_A\, (\bar{S}\ \&\ \bar{Q}))\ \&\ (\neg \bar{Q}\, U_A\, (\neg Q \vee \neg \bar{R}))]$	by DP using 6,7,8,9

The intuition behind step 5 above is seen in the PTL timing diagram of Figure 5.6. The intuition behind final deduction step is seen in Figure 5.7. Using the implication used as the basis for proof rule FGGS, formula (10) of the derivation above can be reduced to

$$\vdash A_{SC} \supset \neg Set_{i+1}\, U\, Set_{i+1}\, U_A \neg \bar{R}_{i+1}.$$

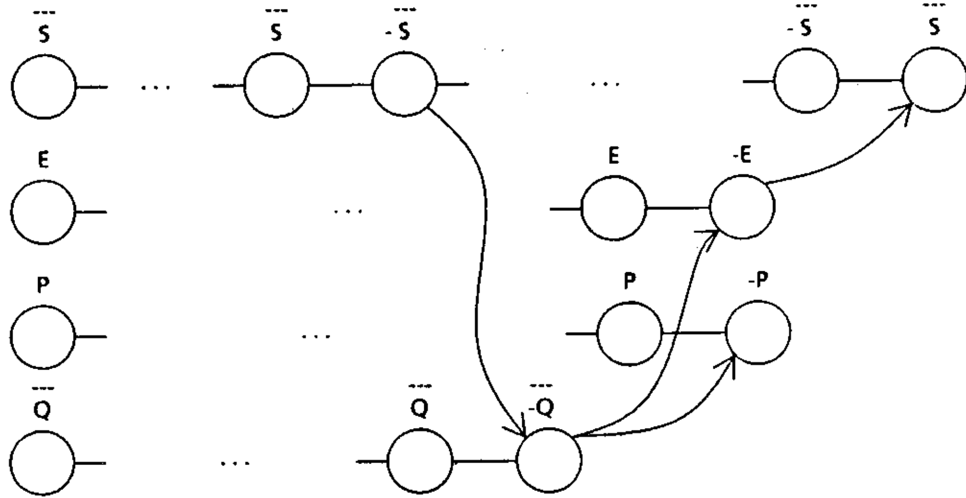


Figure 5.6. PTL Timing Diagram for $\overline{S}_{i+1}$ 1-TSP

Similarly, we derive

$$\vdash \mathrm{A}_{SC} \supset \neg Reset_i \; U \; Reset_i \; U_A \neg \overline{S}_i .$$

We convert the weak U 's into strong U^E 's by using latch liveness, formulas (5.9) and (5.10), and the definition of U^E (T9).

$$\vdash \mathrm{A}_{SC} \supset \neg Set_{i+1} \; U^E \; Set_{i+1} \; U_A \neg \overline{R}_{i+1} \tag{5.17}$$

$$\vdash \mathrm{A}_{SC} \supset \neg Reset_i \; U^E \; Reset_i \; U_A \neg \overline{S}_i \tag{5.18}$$

Now, we turn to the problem of extending the TSPs (5.17) and (5.18) to until $IS(q_{i+1})$. Adding the non-interference properties (5.11) and (5.12) and the condition

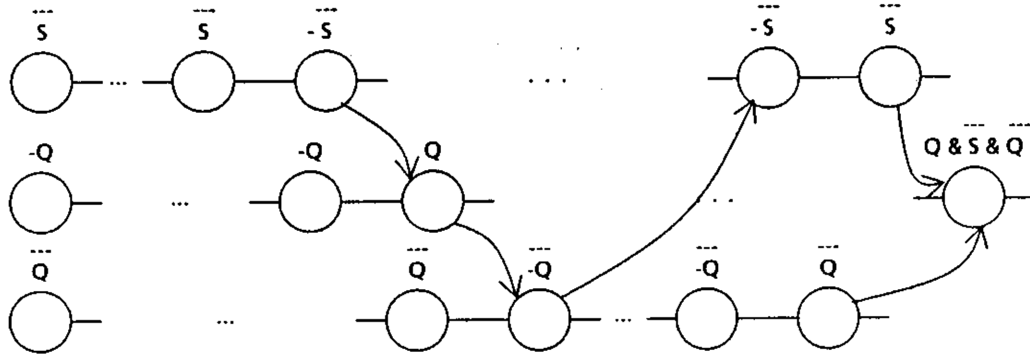


Figure 5.7 PTL Timing Diagram for Set_{i+1} 1-TSP

$$\vdash \mathrm{A}_{SC} \supset \neg IS(q_{i+1})\ U\ (Reset_i\ \&\ Set_{i+1}) \tag{5.19}$$

which comes from the definition of IS, using T25, we have the situation depicted in Figure 5.8, from which (5.15) and (5.16) can be deduced using the DP.

We now have the required TSPs for the active cells. For the inactive cells, $Cell_j$ where $j = i+2, i+3, \cdots, i$, we need the safety property

$$\vdash \Lambda_{SC} \supset Reset_j\ U_A\ IS(q_{i+1}). \tag{5.20}$$

This is derived as follows.

$\vdash \mathrm{A}_{SC} \supset Reset_j\ U_A\ \neg \bar{S}_j$ by $LatchStayReset(j)$

$\vdash \mathrm{A}_{SC} \supset \bar{S}_j\ U_A\ P_j$ from gate axiom by T28

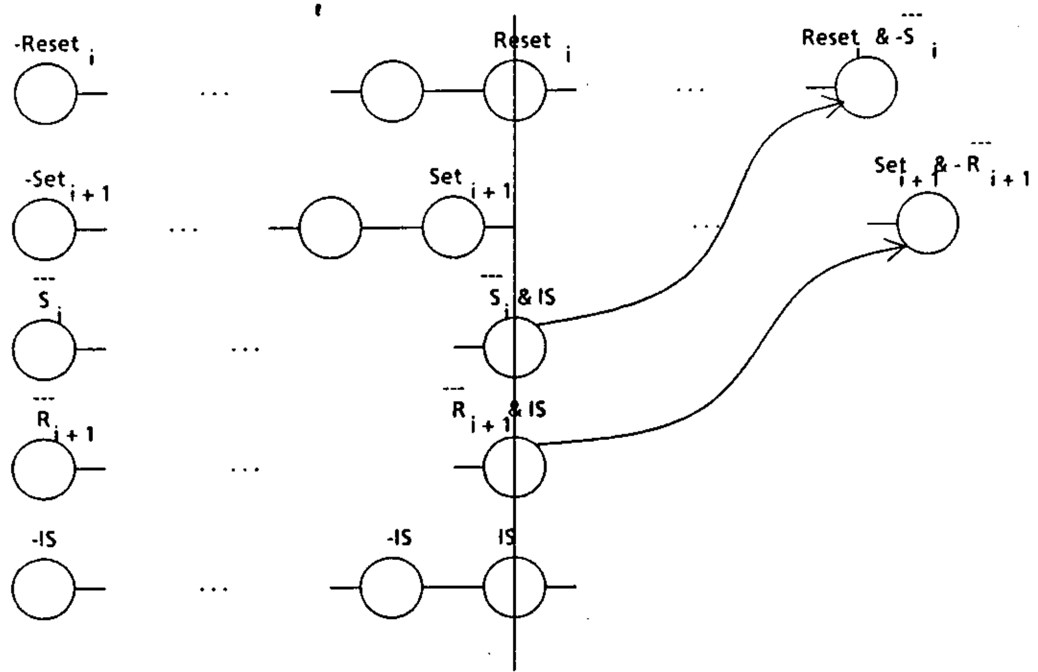


Figure 5.8 PTL Timing Diagrams for $Reset_i$ and Set_{i+1} 1-TSPs

$\vdash A_{SC} \supset \neg P_j \; U_A \; IS(q_{i+1})$ by def. of A_{SC} and T28

$\vdash A_{SC} \supset Reset_j \; U_A \; IS(q_{i+1})$ by BR twice

We also need the property that the $\bar{S}$ lines for each $Cell_j$ remain high through the state change.

$\vdash A_{SC} \supset \neg P_j \; U_A \; IS(q_{i+1})$ by def. of A_{SC} and T28

$\vdash A_{SC} \supset \bar{S}_j \; U_A \; (\neg P_j \; \& \; \neg E_j)$ gate axiom

$\vdash A_{SC} \supset \bar{S}_j \; U_A \; IS(q_{i+1})$ by BR using (1),(2)

Using this result and formulas (5.15), (5.16), (5.20) and a variation of formula (5.19):

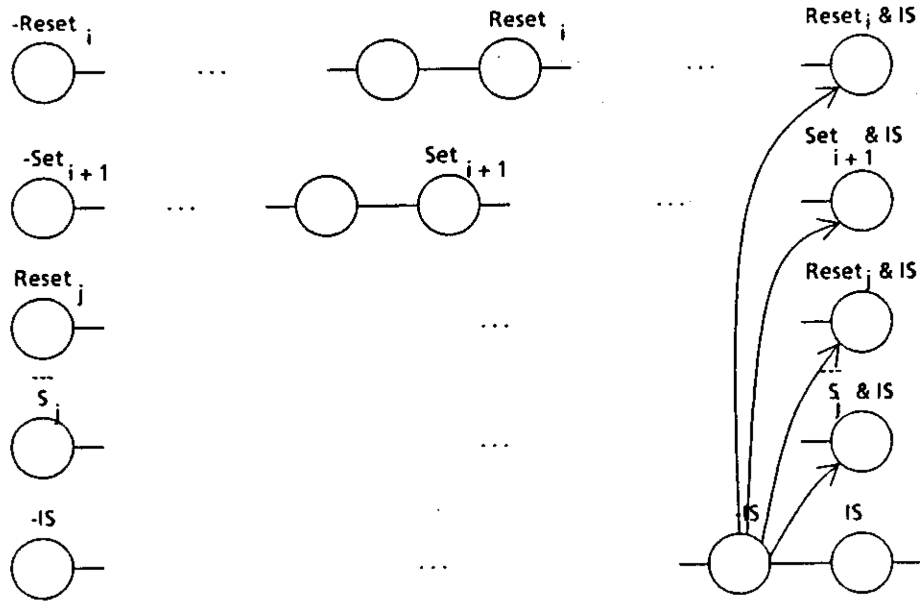


Figure 5.9 PTL Timing Diagram for Controller Liveness Goal 2

$$\vdash \mathrm{A}_{SC} \supset \neg IS(q_{i+1})\ U\ (Reset_i\ \&\ Set_{i+1}\ \&\ Reset_j\ \&\ \bar{S}_j)$$

which comes from the definition of IS using T25 the DP can be used to derive the first of our two liveness goals, formula (5.13). The intuition is provided in Figure 5.9. We now turn to proving the second goal (5.14).

The situation $AIS(q_{i+1})$ is stable as long the next input is not applied. Thus the controller remains in $AIS(q_{i+1})$ until *SetLinesSettled*, i.e., the $\bar{S}_{i+1}$ line goes high again. With the assumption of our second goal, we are able to retain the non-interference properties, since they were proved from I' and gate axioms, both of which are still in effect. Therefore

$$\vdash A_2 \supset \bar{S}_i \; U_A \; IS(q_{i+1}) \tag{5.21}$$

$$\vdash A_2 \supset \bar{R}_{i+1} \; U_A \; IS(q_{i+1}) \tag{5.22}$$

where A_2 is the antecedent of goal (5.14). We use these non-interference properties to derive the fact that the latches of $Cell_i$ and $Cell_{i+1}$ are stable after they undergo their transitions. Since A_2 implies $LatchStayReset_i$, we know that

$$\vdash A_2 \supset Reset_i \; U_A \; \neg \bar{S}_i$$

Combining this with non-interference formula (5.21) using BRyields

$$\vdash A_2 \supset Reset_i \; U_A \; IS(q_{i+1}) \tag{5.23}$$

and, similarly for Set_{i+1},

$$\vdash A_2 \supset Set_{i+1} \; U_A \; IS(q_{i+1}). \tag{5.24}$$

$n-2$ other safety formulas must be proved, one for each of the non-active cells. For $j = i+2, i+3, \ldots, i$, we derive

$$\vdash A_2 \supset (Reset_j \; \& \; \bar{S}_j) \; U_A \; IS(q_{i+1}). \tag{5.25}$$

$\vdash A_2 \supset Reset_j \; U_A \; \neg \bar{S}_j$	by $LatchStayReset_j$
$\vdash A_2 \supset \bar{S}_j \; U_A \; (E_j \; \& \; P_j)$	gate axiom
$\vdash A_2 \supset \neg P_j \; U_A \; IS(q_{i+1})$	by def. of I'
$\vdash A_2 \supset (Reset_j \; \& \; \bar{S}_j) \; U_A \; IS(q_{i+1})$	by GBR

The one line remaining is the only one undergoing a transition, $\bar{S}_{i+1}$. This line is

pulled high by Q_i of the latch in $Cell_i$.

$\vdash (\neg Q \vee \neg P)_{i+1} \twoheadrightarrow \bar{S}_{i+1}$ gate axiom

$\vdash A_2 \supset \neg Q_i$ by PC

$\vdash A_2 \supset \neg Q_i \vee \neg P_{i+1}$ by PC

$\vdash A_2 \supset \nabla \bar{S}_{i+1}$ by trans. and $\Box$ E

We combine formulas (5.23), (5.24), and (5.25) to obtain

$$\vdash A_2 \supset \left[Set_{i+1} \;\&\; \bigwedge_{j=i+2}^{l} (Reset_j \;\&\; \bar{S}_j) \right] U_A \; IS(q_{i+1})$$

This is

$$\vdash A_2 \supset AIS(q_{i+1}) \; U_A \; IS(q_{i+1})$$

saying the circuit will wait for $\bar{S}_{i+1}$ to go high. Since

$$\vdash A_2 \supset \nabla \bar{S}_{i+1},$$

$\bar{S}_{i+1}$ *will* go high, and

$$\vdash A_2 \supset \nabla IS(q_{i+1})$$

is derivable. This last step is proved by the DP using the additional condition $\neg IS(q_{i+1}) \; U \; \bar{S}_{i+1}$ derived from $\Box(IS(q_{i+1}) \supset \bar{S}_{i+1})$ by T25 using the definition of IS. The intuition in the form of a timing diagram is presented in Figure 5.10.

5.4 Discussion

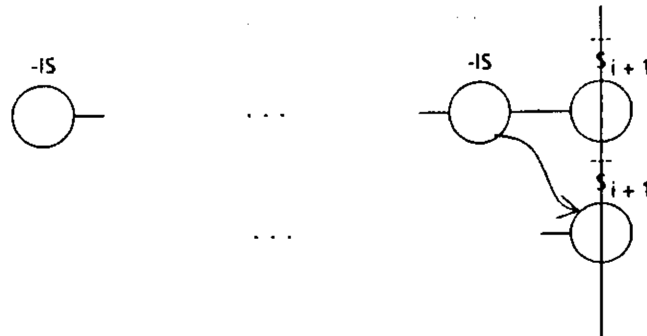


Figure 5.10. PTL Timing Diagram for Controller Liveness Goal 2

Global-time correctness involves quite a number of U -properties. The reason for this is seen in the small example of Figure 5.3. Even for a two-gate circuit it was necessary to specify that the input Z was high long enough for it to synchronize with T to account for the stage synchronization problem. When carrying this out for a conventional size circuit, the number of U -properties grows more than linearly in the number of gates, due to all the gate-to-gate inter connections. The stage sychronization problem takes its toll in the complexity of the proof.

In the previous chapter we were restricted to single transitions and steady-state *behavior*. This eliminated the stage synchronization problem, and we were able to prove every proof rule and theorem by hand. But global-time proving involves so many complex U -properties that the timing diagrams and the DP must be used.

The framework of Chapter 4 and this chapter is a *syntax-directed* one where gates are composed to form circuits, and proofs are contructed in a bottom-up manner along with the circuit contruction. Little attention is paid to the behavior of the circuits and subcircuits except in the PTL timing diagrams. In the next chapter, we consider a different approach where the behavior is derived first, then properties are proved using the information about the behavior.

CHAPTER 6

Execution Graph Method

Rather than directly proving the desired circuit properties as we have done so far, we now consider a proof method, called the Execution Graph Method, where a graph is first constructed then used as a tool to prove properties. The graph is somewhat like a finite state graph where every vertex is associated with a propositional logic formula. Every path through the graph has a corresponding PTL formula dictating the allowable execution sequence. Construction of the graph is an investment which is worthwhile when several properties need to be proved.

6.1 Execution Graphs

Like a program, a circuit has an execution which defines the states in which the circuit may be and the order in which the states are entered. An execution graph is a structure from which the allowable circuit executions can be inferred.

Definition 6.1. A *state template* for a circuit with input, output, and internal line variables $v_1, \cdots, v_l$ is a set of PC formulas

$$\hat{q}_1 \equiv_{def} f_1(v_1, \cdots, v_l)$$

$$\cdots$$

$$\hat{q}_n \equiv_{def} f_n(v_1, \cdots, v_l)$$

involving the line variables such that no two of the $\hat{q}_i$ are true at the same time. Thus it must be true that $\Box(\sum_{i=1}^{n} \hat{q}_i = 1)$.

Definition 6.2. A state template $\hat{q}_1, \cdots, \hat{q}_n$ for a circuit is said to *cover the execution* if it is true that

$$\Box(\hat{q}_1 \vee \cdots \vee \hat{q}_n).$$

This is an essential property of state templates. Only state templates with this property are of interest here.

Definition 6.3. An *execution graph (EG)* for a circuit with a state template $\hat{q}_1, \cdots, \hat{q}_n$ is a pair (V, S) where V is a set of vertices $q_1, \cdots, q_n$ corresponding to the formulas of the state template, and S is the set of *safety arcs* $S \subseteq V \times V - \{<q,q> \mid q \in V\}$. $S(q)$, the set of successor states for state q is defined

$$S(q) = \{q' \mid <q,q'> \in S\}.$$

Note that q cannot have itself as a successor by the definition of S. An EG is a graph representation of the set of safety formulas

$$\hat{q}_1 \Rightarrow \bigvee_{q_{j_1} \in S(q_1)} (\hat{q}_1 \; U \; \hat{q}_{j_1})$$

$$\cdots$$

$$\hat{q}_n \Rightarrow \bigvee_{q_{j_n} \in S(q_n)} (\hat{q}_n \; U \; \hat{q}_{j_n})$$

where $\hat{q}_i$ is true when execution is at vertex q_i.

Each formula above says that if execution is in state q, then it will remain so until execution goes into a state in $S(q)$. q with safety arcs (single arrows) to each of its possible successors $q_1', \cdots, q_k'$ is depicted in Figure 6.1. The name *safety arc* refers to the safety property that the circuit either remains in q forever or switches to one of the successors at some point in time without any intervening undefined or unexpected behavior. In other words, nothing "bad" happens at q until q is left, at which time the responsibility of safety is tranferred to the successor.

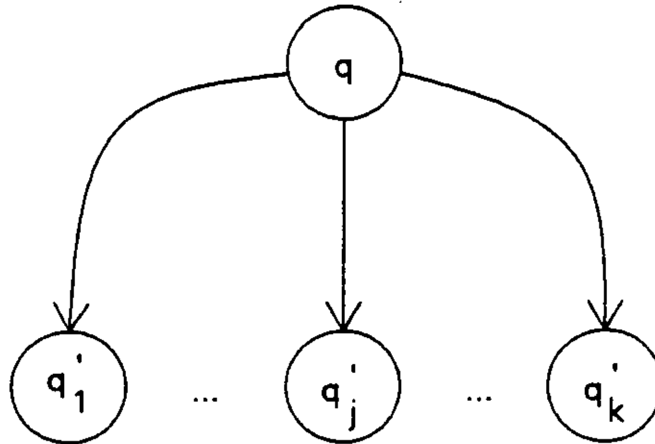


Figure 6.1. An EG Vertex with Safety Arcs

Our attention now turns to execution graphs which not only dictate the allowable states through which execution may go, but also which states are transient and must be exited after some amount of time. These graphs are called safety/liveness execution graphs, since eventualities are present also.

Definition 6.4. A *safety/liveness execution graphs (SLEG)* for circuit with a state template $\hat{q}_1, \cdots, \hat{q}_n$ is a triple (V, S, L) where V and S are the set of vertices and safety arcs, respectively, and $L \subseteq S$ is a set of *liveness arcs.* A SLEG is a graph representation of the set of safety and liveness formulas

$$\hat{q}_1 \Rightarrow \bigvee_{q_{j_1} \in S(q_1)} (\hat{q}_1 \, \zeta \, \hat{q}_{j_1})$$

$$\cdots$$

$$\hat{q}_n \Rightarrow \bigvee_{q_{j_n} \in S(q_n)} (\hat{q}_n \, \zeta \, \hat{q}_{j_n})$$

where each ζ is either weak U, in the case of a safety formula, or strong U^E, in the case of a liveness formula. If a term is of the form $(\hat{q}_i \; U \; \hat{q}_{j_i})$ then $<q_i, q_{j_i}> \in S$, but it is not true that $<q_i, q_{j_i}> \in L$ since the arc is not a liveness arc. If a term is of the form $(\hat{q}_i \; U^E \; \hat{q}_{j_i})$ then $<q_i, q_{j_i}> \in L$. The n PTL formulas associated with the n SLEG vertices are called *SLEG-formulas.*

Since $L \subseteq S$ each liveness arc is also a safety arc. A liveness arc is a special kind of safety arc. Whereas the state transition *may* occur for a safety arc, it *must* occur for a liveness arc.

When a SLEG is constructed, it may not always contain all known information. In particular, a vertex, which has a mix of safety and liveness arcs extending from it to successors, may be able to have the safety arcs transformed into liveness arcs. This may be helpful or even necessary when proving liveness properties of the circuit, which are stronger assertions involving U^E.

Theorem 6.1 (Arc Transformation). If a SLEG vertex q has a safety arc $<q, q_j'>$, and it is known that $\vdash \hat{q} \supset \nabla \neg \hat{q}$ then the safety arc may be transformed into a liveness arc.

Proof Figure 6.2 shows the situation. Vertex q has n successors $q_1', \cdots, q_k'$ with q_j', $1 \le j \le n$ a safety arc. The other successors are either safety (U) or liveness (U^E) arcs. The SLEG-formula corresponding to the vertex q is

$$\hat{q} \Rightarrow \left[\bigvee_{i=1, i \neq j}^{n} (\hat{q} \; \zeta \; q_i') \right] \vee (\hat{q} \; U \; \hat{q}_j')$$

Expanding the second term (by T10) yields

$$\hat{q} \Rightarrow \left[\bigvee_{i=1, i \neq j}^{n} (\hat{q} \; \zeta \; \hat{q}_i') \right] \vee (\hat{q} \; U^E \; \hat{q}_j') \vee \Box \hat{q} \tag{6.1}$$

which allows the possibility of $\Box \hat{q}$. If $\vdash \hat{q} \supset \nabla \neg \hat{q}$ then $\vdash \hat{q} \twoheadrightarrow \neg \hat{q}$. Conjuncting this with formula (6.1) above and reducing eliminates the disjunct $\Box \hat{q}$ because $\Box \hat{q} \; \& \; \nabla \neg \hat{q}$ is false. This leaves

$$\hat{q} \Rightarrow \left[\bigvee_{i=1, i \neq j}^{n} (\hat{q} \; \zeta \; \hat{q}_i') \right] \vee (\hat{q} \; U^E \; \hat{q}_j')$$

the situation in Figure 6.3. •

In summary, if a circuit has a SLEG (V, S, L) where $V = \{q_1, \cdots, q_n\}$ and q_1 is the initial state, then the following set of PTL formulas describe the execution of the circuit beginning with state q_1.

$$\hat{q}_1$$

$$\textit{for each safety arc} \quad <q, q'>, \; \hat{q} \Rightarrow \hat{q} \; U \; \hat{q}'$$

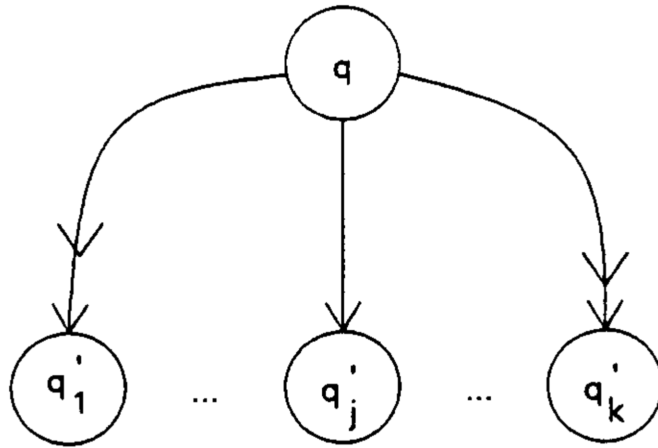


Figure 6.2. Vertex with a Safety Arc before Transformation

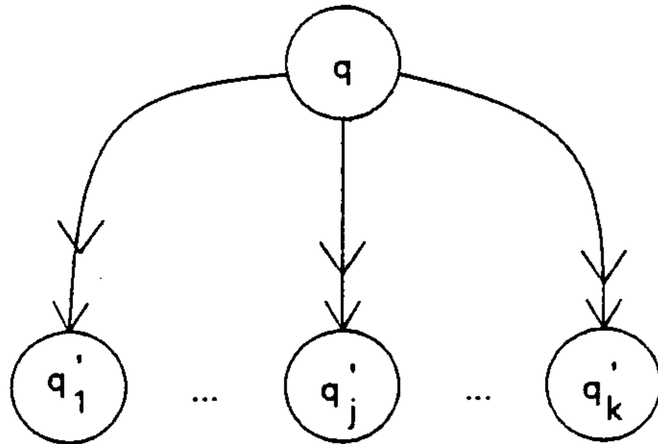


Figure 6.3. Vertex After Transformation

for each liveness arc $<q,q'>$, $\hat{q} \Rightarrow \hat{q}\ U^E\ \hat{q}'$

Theorem 6.2. (Execution Covering). If the following SLEG-formulas for a circuit with state template $\hat{q}_1, \cdots, \hat{q}_n$ are derivable

$$\hat{q}_1 \Rightarrow \bigvee_{q_{j_1} \in S(q_1)} (\hat{q}_1 \; U \; \hat{q}_{j_1})$$

$$\cdots$$

$$\hat{q}_n \Rightarrow \bigvee_{q_{j_k} \in S(q_n)} (\hat{q}_n \; U \; \hat{q}_{j_n}),$$

and $\hat{q}_1$ is derivable, then

$$\vdash \Box (\hat{q}_1 \vee \cdots \vee \hat{q}_n).$$

and the execution is covered by $\hat{q}_1, \cdots, \hat{q}_n$. We call $\hat{q}_1, ..., \hat{q}_n$ a *closed set* of SLEG-formulas.

Proof The proof is by induction on t, the time index of the desired formula: $\bigwedge_{t=0}^{\infty} (\hat{q}_1 \vee \cdots \vee \hat{q}_n)$. For the basis $t = 0$. At time 0, $\hat{q}_1$ is true so $(\hat{q}_1 \vee \cdots \vee \hat{q}_n)$ is true. For the induction step, assume at time t that $(\hat{q}_1 \vee \cdots \vee \hat{q}_n)$ is true. Then we have n identical cases indexed by i, depending upon which $\hat{q}_i$ is true. Let $S(q') = \{q_{j_1}, \cdots, q_{j_k}\}$, such that $k = \mid S(q') \mid$, and let $q' = q_i$. Then the formula

$$\hat{q}' \Rightarrow \left[(\hat{q}' \; U \; \hat{q}_{j_1}) \vee \cdots \vee (\hat{q}' \; U \; \hat{q}_{j_k}) \right]$$

can be transformed, by the definition of U (A8) into

$$\hat{q}' \Rightarrow [\, (\hat{q}_{j_1} \vee (\hat{q}' \;\&\; \bigcirc(\hat{q}' \; U \; \hat{q}_{j_1})))$$

$$\vee \cdots \vee$$

$$(\hat{q}_{j_k} \vee (\hat{q}' \;\&\; \bigcirc(\hat{q}' \; U \; \hat{q}_{j_k})))\,]$$

Since $\hat{q}'$ is true, all the $\hat{q}_i$ are mutually exclusive (by definition 6.1), and none of the

$\hat{q}_i$ are q' by definition of $S(q')$, we know that all of $q_{j_1}, \cdots, q_{j_k}$ are false. The first of the two disjuncts in each of the k disjuncts above, $\hat{q}_{j_i}$, is eliminated, leaving the second disjunct in each case.

$$\hat{q}' \Rightarrow [\ \bigcirc(\hat{q}'\ U\ \hat{q}_{j_1})$$

$$\vee \ldots \vee$$

$$\bigcirc(\hat{q}'\ U\ \hat{q}_{j_k})\ ].$$

Expanding again using the definition of U yields

$$\bigcirc[(\hat{q}_{j_1} \vee \cdots \vee \hat{q}_{j_k}) \vee \hat{q}'].$$

Note that all the above $k+1$ $\hat{q}$ are selected from the state templates $\hat{q}_1, \cdots, \hat{q}_n$. Therefore $\bigcirc(\hat{q}_1 \vee \cdots \vee \hat{q}_n)$ is true at time t. It is then true that $(\hat{q}_1 \vee \cdots \vee \hat{q}_n)$ at time $t+1$, which is the induction hypothesis for time $t+1$. •

The set of SLEG-formulas is "closed" in the sense that at no point in time is it possible for the execution be outside the scope of the state template. If it is possible to construct a state template so that the SLEG-formulas may be derived in the PTL deductive system, then Theorem 6.2 says that the state template covers the execution and is considered satisfactory. A bad state template is one for which SLEG-formulas cannot be derived.

Consider the simple two-inverter delay element of Figure 6.4. Assume the usual inverter gate axioms and the additional input contraint that the input must wait for the output to settle through the inverters.

$$\vdash A \Rightarrow A\ U_A\ C$$

$$\vdash \neg A \Rightarrow \neg A \; U_A \; \neg C$$

If the state template

$$\hat{q}_1 \equiv_{def} A \;\&\; \neg B \;\&\; C$$

$$\hat{q}_2 \equiv_{def} \neg A \;\&\; \neg B \;\&\; C$$

$$\hat{q}_3 \equiv_{def} \neg A \;\&\; B \;\&\; C$$

$$\hat{q}_4 \equiv_{def} \neg A \;\&\; B \;\&\; \neg C$$

is used, only half of the execution is covered. The formulas

$$\vdash \hat{q}_1 \Rightarrow \hat{q}_1 \, U \, \hat{q}_2$$

$$\vdash \hat{q}_2 \Rightarrow \hat{q}_2 \, U \, \hat{q}_3$$

$$\vdash \hat{q}_3 \Rightarrow \hat{q}_3 \, U \, \hat{q}_4$$

$$\vdash \hat{q}_4 \Rightarrow \hat{q}_4 \, U \, \hat{q}_5$$

are derivable from inverter gate axioms for some q_5, but q_5 is not in our state template. In fact, no set of closed SLEG-formulas can be derived for $q_1, \ldots, q_4$, the bad state template. Only a 6-state template will do.

6.2 Checking Specifications

Once the SLEG is constructed, checking the validity of the specifications proceeds quite naturally. The specification is derived from the SLEG-formulas. We partition the types of specifications addressed here into *safety* assertions of the form $\xi \Rightarrow \xi \, U \, \eta$ and *liveness* assertions of the form $\xi \twoheadrightarrow \eta$. A method for checking

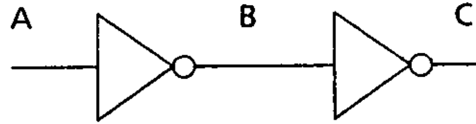


Figure 6.4. A Two-Inverter Delay Element

whether these specification forms hold for a SLEG is presented below.

Definition 6.5. Π, the *set of paths* emanating from a vertex q is defined

$$\Pi(q)=\{<q_1, \cdots, q_k> \mid q_1=q \text{ and } q_{j+1} \in S(q_j),\ 1 \le j \le k\}.$$

By the Definition 6.5, a path through a SLEG is finite but unbounded in length. Since paths may be arbitrarily long, there is an infinite number of paths in any SLEG with a cycle. Therefore Π is possibly infinite. We can eliminate paths which traverse cycles more than one time as it is redundant to repeatedly traverse them when checking whether specifications hold. The set Λ is defined to contain only paths which traverse cycles once.

Definition 6.6. The finite set of *non-repeating paths* emanating from a vertex is defined

$$\Lambda(q)=\{<q_1, \cdots, q_k> \mid <q_1, \cdots, q_k> \in \Pi(q_1)$$

$$\text{and no cycle } q_1', \cdots, q_{m-1}', q_1'$$

$$\text{appears more than once in } q_1, \cdots, q_k\}.$$

Since there are a finite number of cycles, there are a finite number of paths which may contain each cycle at most once. Let $v = |V|$ be the number of vertices. The number of cycles from a vertex q back to itself of length $a \le v-1$ using each of the other vertices is $({}_aP_a)$, where $({}_nP_r)$ is the number of permutations of n elements taken r at a time. The total number of such cycles is

$$c = ({}_{v-1}P_{v-1})({}_{v-1}P_{v-2}) \cdots ({}_{v-1}P_1).$$

The number of non-repeating paths is bounded by b where

$$b = ({}_cP_c)({}_cP_{c-1}) \cdots ({}_cP_1).$$

A path with a repeated cycle of the form

$$<q, \cdots, q_1, \cdots, q_{m-1}, q_1, \cdots, q_{m-1}, q_1, \cdots, q'>$$

with length m loop $q_1, \cdots, q_{m-1}, q_1$ appearing twice is transformed into a path without the repeated cycle

$$<q, \cdots, q_1, \cdots, q_{m-1}, q_1, \cdots, q'>$$

by removing the duplicate vertices $q_2, ..., q_{m-1}, q_1$. A path with a repeated cycle of the form

$$<q, \cdots, q_1, \cdots, q_{m-1}, q_1, \cdots, q_1, \cdots, q_{m-1}, q_1, \cdots, q'>$$

with length m loop $q_1, \cdots, q_{m-1}, q_1$ appearing twice is transformed into a path without the repeated cycle

$$<q, \cdots, q_1, \cdots, q_{m-1}, q_1, \cdots, q_1, \cdots, q'>$$

by removing the duplicate vertices $q_2, ..., q_{m-1}, q_1$.

We also restrict our attention to the longest paths which will demonstrate the properties we expect to hold.

Definition 6.7. The finite set of *maximal non-repeating paths* emanating from a vertex q is defined

$$\Phi(q) = \{<q_1, \cdots, q_k> \mid q_1 = q \textit{ and } <q_1, \cdots, q_k> \in \Lambda(q_1)$$

$$\textit{and no longer non-repeating path } <q_1, \cdots, q_{k'}> \in \Lambda(q_1),$$

$$k'>k \text{ , } \textit{containing } <q_1, \cdots, q_k> \textit{ exists } \}$$

6.2.1 Checking Safety Specifications

We first define what it means for a safety formula to "hold" on a template. We use the $\models$ symbol for "holds on a template."

Definition 6.8. A safety formula $\xi \Rightarrow \xi \; U \; \eta$ *holds* on a template $q_1, \cdots, q_k$, written

$$\{q_1, \cdots, q_k\} \models \xi \Rightarrow \xi \; U \; \eta,$$

if and only if

$$\forall q \in \{q_1, \cdots, q_m\}. \; [\hat{q} \supset \xi] \supset \forall <q_1', \cdots, q_m'> \in \Phi(q).$$

$$<q_1', \cdots, q_m'> \models \xi \; U \; \eta$$

where

$$<q_1, \cdots, q_k> \models \xi \; U \; \eta \;\; \equiv_{def} \;\; \exists q_l \in \{q_1, \cdots, q_k\}. \; \hat{q}_l \supset (\eta \vee \Box \xi)$$

$$\& \left[\bigwedge_{j=1}^{l-1} [\hat{q}_j \supset \xi] \right]$$

Lemma 6.1. If $\xi \, U \, \eta$ holds on a repeating path then it holds on the path's non-repeating counterpart.

Proof Consider a repeating path $<q_1, \cdots, q_k>$. If $\xi \, U \, \eta$ holds on it then it is because there is an l such that $1 \le l \le k$ where $\hat{q}_l \supset (\eta \vee \Box \xi)$ and $\hat{q}_j \supset \xi$ for each j such that $1 \le j < l$. We have any or all of the following three cases depending upon where the vertices are removed in the non-repeating path.

If elimination of repetition removes vertices after q_l, then that still leaves q_l which implies $(\eta \vee \Box \xi)$ and all the (0 or more) q_j vertices which imply ξ so that $\xi \, U \, \eta$ holds on the new path.

If elimination of repetition removes q_l, then there must be another occurrence of q_l further leftward in the path, otherwise q_l would not be eliminated. Thus there is another q_l satisfying $q_l \supset (\eta \vee \Box \xi)$. The q_j vertices which imply ξ are still there from the original $q_1, \cdots, q_l$. Thus $<q_1, \cdots, q_k> \models \xi \, U \, \eta$.

If some of the q_j vertices to the left of q_l are eliminated, then q_l still remains such that $\hat{q}_l \supset (\eta \vee \Box \xi)$ and the remaining q_j still imply ξ. Thus $\xi \, U \, \eta$ holds on the new path. •

Lemma 6.2. If $\xi \, U \, \eta$ hold on a non-maximal non-repeating path then it holds on the path's maximal non-repeating counterpart.

Proof Consider a non-maximal path $<q_1, \cdots, q_k>$. Let its maximal counterpart be $<q_1, \cdots, q_{k'}>$ where $k' \ge k$. If there exists a q_l which implies $(\eta \vee \Box \xi)$ and an appropiate sequence $q_1, \cdots, q_{l-1}$ where ξ is implied in the shorter path then clearly the longer path has the same q_l and q_j. So $<q_1, \cdots, q_{k'}> \models \xi \, U \, \eta$. •

Theorem 6.3 (Safety Checking). If $\{q_1, \cdots, q_n\} \models \xi \Rightarrow \xi\, U\, \eta$ for a circuit whose execution is covered by state template $\hat{q}_1, \cdots, \hat{q}_n$ then $\vdash \xi \Rightarrow \xi\, U\, \eta$.

Proof The definition of $\{q_1, \cdots, q_n\} \models \xi \Rightarrow \xi\, U\, \eta$ says to consider each $q_i \in \{q_1, \cdots, q_n\}$ separately. For each q_i we have, say, p_i maximal non-repeating paths in $\Phi(q_i)$. Lemmas 6.1 and 6.2 allow us to only consider maximal non-repeating paths since if $\xi\, U\, \eta$ holds on any path, it will hold on its maximal non-repeating counterpart. By the definition of $<q_1', \cdots, q_k'> \models \xi\, U\, \eta$ there is a q_l' in this path where $\hat{q}_l' \supset (\eta \vee \Box \xi)$ and $\hat{q}_j' \supset \xi$ for each j such that $1, \cdots, l-1$. So for path $<q_1', \cdots, q_l'>$ there must be a set of $l-1$ SLEG-formulas

$$\vdash \hat{q}_1' \Rightarrow (\hat{q}_1'\, U\, \hat{q}_2') \vee w_1$$

$$\cdots$$

$$\vdash \hat{q}_{l-1}' \Rightarrow (\hat{q}_{l-1}'\, U\, \hat{q}_l') \vee w_{l-1}$$

We can eliminate the w_j since they are true for paths other than the one under consideration. From the $l-1$ SLEG-formulas we can derive

$$\vdash \hat{q}_1' \Rightarrow \hat{q}_1'\, U\, \cdots\, U\, \hat{q}_{l-1}'\, U\, \hat{q}_l'$$

by

$$\models (w_1 \Rightarrow w_1\, U\, w_2)\ \&\ \cdots\ \&\ (w_{n-1} \Rightarrow w_{n-1}\, U\, w_n) \supset$$

$$w_1 \Rightarrow w_1\, U\, w_2\, U\, \cdots\, U\, w_n,$$

(proved on the DP) where q_l' is the vertex which implies $(\eta \vee \Box \xi)$. By T15 then

$$\vdash \hat{q}_1' \Rightarrow \xi\, U\, \cdots\, U\, \xi\, U\, (\eta \vee \Box \xi)$$

and therefore

$$\vdash \hat{q}_1' \Rightarrow \xi\, U\, (\eta \vee \Box \xi)$$

by T27. Since $\vdash \xi\, U\, (\eta \vee \Box \xi) \supset (\xi \supset \xi\, U\, (\eta \vee \Box \xi))$ from PC we know $\vdash \xi\, U\, (\eta \vee \Box \xi) \Rightarrow (\xi \supset \xi\, U\, (\eta \vee \Box \xi))$ by $\Box I$ and so, by modus ponens, $\vdash \hat{q}_1' \Rightarrow (\xi \supset \xi\, U\, (\eta \vee \Box \xi))$.

The same deduction process is used for each of the p maximal non-repeating paths from q to convert

$$\begin{aligned} \vdash \hat{q}' \Rightarrow\ & \hat{q}'\, U\, \cdots\, U\, \hat{q}_{k_1}' \\ & \vee \ldots \vee \\ & \hat{q}'\, U\, \cdots\, U\, \hat{q}_{k_p}' \end{aligned}$$

into

$$\vdash \hat{q}' \Rightarrow (\xi \supset \xi\, U\, (\eta \vee \Box \xi)) \vee \cdots \vee (\xi \supset \xi\, U\, (\eta \vee \Box \xi))$$

and into

$$\vdash \hat{q}' \Rightarrow (\xi \supset \xi\, U\, (\eta \vee \Box \xi)) \tag{6.2}$$

for every vertex q' in the state template. Since the state template covers the execution,

$$\vdash \Box (\hat{q}_1 \vee \cdots \vee \hat{q}_n)$$

holds. Then for each of the n vertices, we apply the entailment (6.2) above yielding $\vdash \xi \Rightarrow \xi\, U\, (\eta \vee \Box \xi)$. We then split the formula using ER and

$$\models w_1 \, U \, (w_2 \vee w_3) \equiv (w_1 \, U \, w_2) \vee (w_1 \, U \, w_3)$$

(Theorem 24 of [Man81a]) into

$$\vdash (\xi \supset (\xi \, U \, \eta)) \vee (\xi \supset \xi \, U \, \Box \xi).$$

The second disjunct can be reduced to

$$\xi \supset \Box \xi$$

by ER and the implication

$$\models (\xi \, U \, \Box \xi) \supset \Box \xi,$$

(proved on the DP). The entire formula becomes (using T11) $\xi \supset \xi \, U \, \eta$. Then, by PTL inference rule R3,

$$\vdash \xi \Rightarrow \xi \, U \, \eta$$

•

6.2.2 Checking Liveness Specifications

When checking liveness properties, the notion of a "live" path occurs. A liveness formula is true on a path only if the path is live from the initial vertex to the vertex where the desired eventuality occurs.

Definition 6.9. A *live path* is a path which consists of vertices connected by liveness arcs. $<q_1, \cdots, q_k>$ is live if $<q_j, q_{j+1}> \in L$ for each j, $1 \leq j < k$.

We define what it means for a liveness formula to "hold" on a template.

Definition 6.10. A liveness formula $\xi \twoheadrightarrow \eta$ *holds* on a template $q_1, \cdots, q_k$, written

$$\{q_1, \cdots, q_k\} \models \xi \twoheadrightarrow \eta,$$

if and only if

$$\forall q \in \{q_1, \cdots, q_m\}.\ [\hat{q} \supset \xi] \supset \forall <q_1', \cdots, q_m'> \in \Phi(q).$$

$$<q_1', \cdots, q_m'> \models \nabla \eta$$

where

$$<q_1, \cdots, q_k> \models \nabla \eta \equiv_{def} \exists q_l \in \{q_1, \cdots, q_k\}.\ \hat{q}_l \supset \eta$$

$$\textit{and } <q_1, \cdots, q_l> \textit{ is live.}$$

Lemma 6.3. If $\nabla \eta$ holds on a repeating path then it holds on the path's non-repeating counterpart.

Proof Consider a repeating path $<q_1, \cdots q_k>$. If $\nabla \eta$ holds on it then it is because there is an l such that $1 \leq l \leq k$ where $<q_1, \cdots, q_l>$ is live and $\hat{q}_l \supset \eta$. We have any or all of the following cases depending upon which vertices are removed to obtain the non-repeating path.

If elimination of repetition removes vertices after q_l, then $<q_1, \cdots, q_l>$ is still live and $\hat{q}_l \supset \eta$ still.

If elimination of repetition removes q_l itself, then there must have another occurrence of q_l further leftward in the path such that $\hat{q}_l \supset \eta$. Since the path is live from q_1 up to q_l, it will be live from q_1 up to the new q_l.

If some of the vertices to the left of q_l are removed, then the new path is still live and q_l remains.

In each case $\nabla \eta$ holds on the new non-repeating path. •

Lemma 6.4. If $\nabla \eta$ holds on a non-maximal, non-repeating path then it holds on the path's maximal non-repeating counterpart.

Proof Consider a non-maximal path $<q_1, \cdots, q_k>$. Let its maximal counterpart be $<q_1, \cdots, q_{k'}>$, where $k' \geq k$. If there exists a q_l which implies η on live path $<q_1, \cdots, q_l>$ in the shorter path, then clearly the same is true on the longer path. So $<q_1, \cdots, q_k> \models \nabla \eta$. •

Theorem 6.4 (Liveness Checking). If $\{q_1, \cdots, q_n\} \models \xi \twoheadrightarrow \eta$ for a circuit whose execution is covered by state template $\hat{q}_1, \cdots, \hat{q}_n$, then $\vdash \xi \twoheadrightarrow \eta$.

Proof The definition of $\{q_1, \cdots, q_n\} \models \xi \twoheadrightarrow \eta$ says to consider each $q_i \in \{q_1, \cdots, q_n\}$ separately. For each q_i we have, say, p_i maximal non-repeating paths in $\Phi(q_i)$. Lemmas 6.3 and 6.4 allow us to only consider maximal non-repeating paths since if $\nabla \eta$ holds on any path, it will hold on its maximal non-repeating counterpart. By the definition of $<q_1', \cdots, q_{k'}'> \models \nabla \eta$ there is a q_l' in this path where $\hat{q}_l' \supset \eta$ and $<q_1', \cdots, q_l'>$ is live. So for path $<q_1', \cdots, q_l'>$ there must be a set of $l-1$ live SLEG-formulas

$$\vdash \hat{q}_1' \Rightarrow (\hat{q}_1' \, U^E \, \hat{q}_2') \vee w_1$$

$$\cdots$$

$$\vdash \hat{q}_{l-1}' \Rightarrow (\hat{q}_{l-1}' \, U^E \, \hat{q}_l') \vee w_{l-1}$$

We can eliminate the w_j since they are true for paths other than the one under

consideration. From the $l-1$ SLEG-formulas we can derive

$$\vdash \hat{q}_1' \Rightarrow \hat{q}_1' \, U^E \cdots U^E \, \hat{q}_{l-1}' \, U^E \, \hat{q}_l'$$

by

$$\models (w_1 \Rightarrow w_1 \, U^E \, w_2) \,\&\, \cdots \,\&\, (w_{n-1} \Rightarrow w_{n-1} \, U^E \, w_n) \supset$$

$$w_1 \Rightarrow w_1 \, U^E \, w_2 \, U^E \cdots U^E \, w_n,$$

(proved on the DP) where q_l' is the vertex which implies η. By T16 then

$$\vdash \hat{q}_1' \Rightarrow \xi \, U^E \cdots U^E \, \xi \, U^E \, \eta$$

and

$$\vdash \hat{q}_1' \Rightarrow \xi \, U^E \, \eta$$

and therefore

$$\vdash \hat{q}_1' \Rightarrow \nabla \eta$$

by T9. Since $\vdash \nabla \eta \supset (\xi \supset \nabla \eta)$ from PC we know $\vdash \nabla \eta \Rightarrow (\xi \supset \nabla \eta)$ by $\Box I$ and so, by modus ponens, $\vdash \hat{q}_1' \Rightarrow (\xi \supset \nabla \eta)$.

The same deduction process is used for each of the p maximal non-repeating paths from q' to convert

$$\vdash \hat{q}' \Rightarrow \hat{q}' \, U^E \cdots U^E \, \hat{q}_{l_1}'$$

$$\vee \cdots \vee$$

$$\hat{q}' \, U^E \cdots U^E \, \hat{q}_{l_p}'$$

into

$$\vdash \hat{q}' \Rightarrow (\xi \supset \nabla \eta) \vee \cdots \vee (\xi \supset \nabla \eta)$$

and into

$$\vdash \hat{q}' \Rightarrow (\xi \supset \nabla \eta) \tag{6.3}$$

for every vertex q in the state template. Since the state template covers the execution,

$$\vdash \Box(\hat{q}_1 \vee \cdots \vee \hat{q}_n)$$

holds. Then for each of the n vertices, we apply the entailment (6.3) above yielding $\vdash \xi \supset \nabla \eta$ and then, by PTL axiom R3,

$$\vdash \xi \twoheadrightarrow \eta$$

•

6.2.3 An Example

To illustrate the use of the framework just introduced, we now apply the Execution Graph Method to an example. A more ambitious example which more fully displays the method follows in Section 6.4.

The two-inverter delay element of Figure 6.4 (without the input contraint of Section 6.1) has 8 states. The following gate axioms determine a SLEG.

$$\vdash B \Rightarrow B \ U_A \ A$$

$$\vdash \neg B \Rightarrow \neg B \ U_A \ \neg A$$

$$\vdash A \Rightarrow A\ U_A \neg B$$

$$\vdash \neg A \Rightarrow \neg A\ U_A\ B$$

$$\vdash C \Rightarrow C\ U_A\ B$$

$$\vdash \neg C \Rightarrow \neg C\ U_A \neg B$$

$$\vdash B \Rightarrow B\ U_A \neg C$$

$$\vdash \neg B \Rightarrow \neg B\ U_A\ C$$

The state template is

$$\hat{q}_1 \equiv_{def} A \ \&\ \neg B \ \&\ C$$

$$\hat{q}_2 \equiv_{def} \neg A \ \&\ \neg B \ \&\ C$$

$$\hat{q}_3 \equiv_{def} \neg A \ \&\ B \ \&\ C$$

$$\hat{q}_4 \equiv_{def} \neg A \ \&\ B \ \&\ \neg C$$

$$\hat{q}_5 \equiv_{def} A \ \&\ B \ \&\ \neg C$$

$$\hat{q}_6 \equiv_{def} A \ \&\ \neg B \ \&\ \neg C$$

$$\hat{q}_7 \equiv_{def} \neg A \ \&\ \neg B \ \&\ \neg C$$

$$\hat{q}_8 \equiv_{def} A \ \&\ B \ \&\ C.$$

The SLEG-formulas

$$\vdash \hat{q}_1 \Rightarrow \hat{q}_1\ U\ \hat{q}_2$$

$$\vdash \hat{q}_2 \Rightarrow \hat{q}_2\, U\, \hat{q}_3$$

$$\vdash \hat{q}_3 \Rightarrow \hat{q}_3\, U\, (\hat{q}_4 \vee \hat{q}_7 \vee \hat{q}_8)$$

$$\vdash \hat{q}_4 \Rightarrow \hat{q}_4\, U\, \hat{q}_5$$

$$\vdash \hat{q}_5 \Rightarrow \hat{q}_5\, U\, \hat{q}_6$$

$$\vdash \hat{q}_6 \Rightarrow \hat{q}_6\, U\, (\hat{q}_1 \vee \hat{q}_2 \vee \hat{q}_7)$$

$$\vdash \hat{q}_7 \Rightarrow \hat{q}_7\, U\, \hat{q}_2$$

$$\vdash \hat{q}_8 \Rightarrow \hat{q}_8\, U\, \hat{q}_5$$

are derivable from inverter safety axioms, yielding the EG of Figure 6.5 with the values of the variables A, B, and C appearing in the vertices.

A PTL timing diagram can be used to show the situation in each of the 8 states. For example, in state q_1 we have the situation shown in Figure 6.6. There A is free to go low at any time since it is not under any input constraint or restriction from a gate axiom, B must stay low until A goes low because of an inverter output safety axiom, and C must stay high because of another inverter output safety axiom. >From the three signal time lines and the two dependencies we hypothesize that

$$q_1 \Rightarrow q_1\, U\, q_2$$

from the diagram, and we use the DP to convince us of our hypothesis.

The EG should be converted to a SLEG next. Using Arc Transformation (Theorem 6.1) with the known gate liveness axioms, the formulas

$$\vdash \hat{q}_2 \supset \nabla \neg \hat{q}_2$$

$$\vdash \hat{q}_3 \supset \nabla \neg \hat{q}_3$$

$$\vdash \hat{q}_5 \supset \nabla \neg \hat{q}_5$$

$$\vdash \hat{q}_6 \supset \nabla \neg \hat{q}_6$$

$$\vdash \hat{q}_7 \supset \nabla \neg \hat{q}_7$$

$$\vdash \hat{q}_8 \supset \nabla \neg \hat{q}_8$$

are derivable. For example, for q_0,

$\vdash \hat{q}_0 \supset \neg B$

$\vdash \neg B \supset \nabla C$

$\vdash C \supset \neg q_0$ *so* $\vdash \nabla C \supset \nabla \neg q_0$ *by* ∇ ∇ *Rule*

$\vdash q_0 \supset \nabla \neg q_0$ *by* $\supset$ *-transitivity twice*

Thus all arcs except $<q_1,q_2>$ and $<q_4,q_5>$ can become liveness arcs by the Arc Transformation Theorem, as depicted in Figure 6.7. Checking the liveness formula $A \twoheadrightarrow C$ is done by checking all 8 states for whether A is true. It is true in states q_1, q_5, q_6, q_8. For each of these vertices, we check for ∇C on all maximal non-repeating paths beginning at the vertex. For example, from vertex q_8 it is seen that for two live paths $<q_8, q_5, q_6, q_1>$ and $<q_8, q_5, q_6, q_7, q_2, q_3, q_8>$, ∇C holds. Using the Liveness Checking Theorem $\vdash A \twoheadrightarrow C$.

6.3 Generation of SLEGs

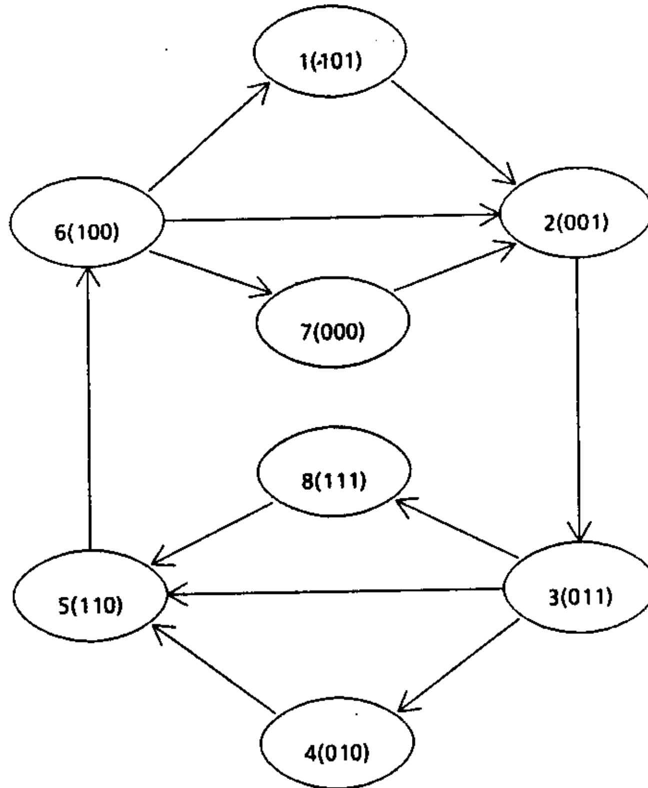


Figure 6.5. EG for Two-Inverter Delay

Up to now SLEGs have been constructed by hand from the set of assumptions, the gate or module axioms and input contraints, using the PTL theorems and PTL timing diagrams. Their main contribution to the proof process is that they are a provably correct set of intermediate formulas from which the desired specifications can be proved. The SLEGs are a way of organizing known information into a form which can be used by the algorithms presented for Safety Checking and Liveness Checking.

We introduce two algorithms for Safety Checking an Liveness Checking (given in Theorems 6.3 and 6.4), we have not discussed and algorithm for contructing SLEGs. Using the algorithm presented here, the SLEG can be automatically generated from the set of assumptions.

The algorithm tries to cover the execution of the circuit with a minimal number of vertices. It begins with the trivial 1-vertex sets, then tries 2-vertex sets, and so on

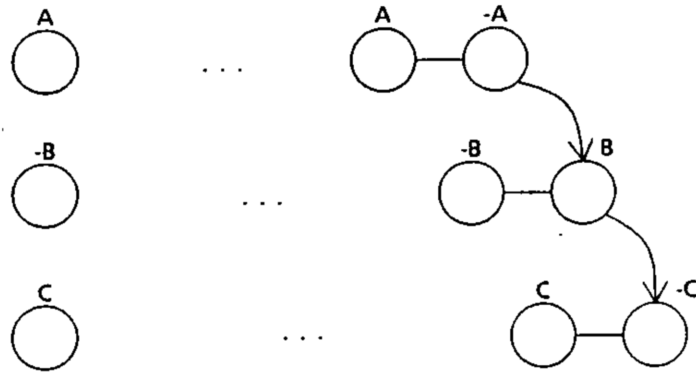


Figure 6.6. PTL Timing Diagram for Initial State q_1

until a EG which has a corresponding set of derivable SLEG-formulas and, hence covers the execution, can be constructed. The DP is consulted for every vertex in a candidate EG. If the DP says that the SLEG-formula corresponding to the vertex is not valid, then the EG is eliminated as a candidate. If all the SLEG-formulas for the vertices are approved by the DP, then the graph covers the execution, and we accept it as our answer.

To derive the SLEG-formula for a vertex of a candidate SLEG, we use the assumptions set forth at the outset (gate or module axioms and input constraints.) The algorithm tries all combinations of successors for each vertex which are in the innermost two loops. For example, if a candidate SLEG consists of vertices q_1, q_2, and q_3 and we are currently trying to discover the SLEG-formula for vertex q_1, we try all possible the successor sets $S(q_1)$ which are $\{\}$, $\{q_2\}$, $\{q_3\}$, and $\{q_2,q_3\}$ by

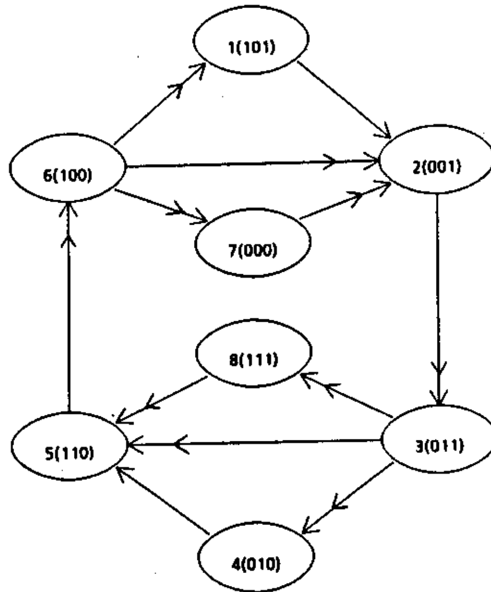


Figure 6.7. SLEG for Two-Inverter Delay

constructing the various EGs containing the appropriate arcs. In general, it will be the case that only one of the successor sets are correct or none are correct in which case the candidate EG is discarded.

The algorithm appears below.

begin with state template $\hat{q}_1, \cdots, \hat{q}_n$.

foundSLEG := **false**;

forall i = 1 **to** 2^l **do** /* for l lines in circuit */

    forall i-state sets $q_1, \cdots, q_i$ **do**

    1: **forall** EGs formed from $q_1, \cdots, q_i$ **do**

        forall vertices q in EG **do**

            submit formula corresponding to q to DP;

            if DP returns not valid **then**

```
                leave 1;
            end;
            use Arc Transformation to convert safety arcs to liveness arcs;
            foundSLEG := true;
            exit;
        end;
    end;
end.
```

6.4 Application to Self-Timed Circuits

Self-timed circuits operate for an indefinite amount of time and send completion signals when done. Unger [Ung69, Arm69] discusses the design of asynchronous circuits with completion signals. Seitz [Sei80a, Sei80b] discusses self-timed VLSI circuits with control signalling embedded in double-rail codes. As mentioned in Section 3.3.1 above, double rail codes use two wires labelled x^1 and x^0 to encode the three values $\{0, 1, \lambda\}$ on a wire x where λ is undefined. Malachi and Owicki write specifications for several of these circuits, including a state machine and pipeline element [Mal81]. We will consider their pipeline element PL which is depicted in Figure 6.8.

6.4.1 Self-Timed Protocol

The "definedness" predicates d and D are defined over logic variables x and variable sets X and Y as

$$d(X) \equiv_{def} (\exists x \in X).d(x) \;\; and \;\; D(X) \equiv_{def} (\forall x \in X).d(x)$$

where

$$d(x) \equiv_{def} x^1 \vee x^0$$

That is, variable x is defined if either wire associated with x is high. In addition, the properties

$$d(X \cup Y) \equiv d(X) \vee d(Y) \;\; and \;\; D(X \cup Y) \equiv D(X) \;\&\; D(Y).$$

hold.

Self-timed circuits operate as follows. The initial state for most signals is usually all-undefined ($\neg\, d$). Inputs start becoming defined ($d(I)$), and the circuit begins to react to the inputs. After some time all inputs are defined ($D(I)$) and this situation is a signal for the circuit to evenutally complete its response with all the outputs defined ($D(O)$). After all outputs are defined the next input can be presented. To distinguish between consecutive inputs, a spacer input of all-undefined is used. The input becomes not all-defined ($\neg\, D(I)$) and then all-undefined ($\neg\, d(I)$) once again.

The output reacted to the all-undefined input by eventually becoming all-defined ($D(O)$). It similarly reacts to the spacer by eventually becoming all-undefined ($\neg\, d(O)$). Thus a set of signals generally obey the transitions

$$\neg\, d \rightarrow d \rightarrow D \rightarrow \neg\, D \rightarrow \neg\, d$$

6.4.2 Module Axioms

Rather than using gate axioms as our set of given assumptions, we use *module axioms* which are formulas axiomatizing the behavior of the modules constituting PL. Malachi and Owicki give the following axioms for the combinational logic module

(CL), with the stronger U_A here in place of U . CL is the computational part of PL and not a control module like the align and converter (C).

$$\phi_{CL} \vdash \neg d(B) \Rightarrow \neg d(B)\, U_A\, d(I) \tag{CL1}$$

$$\phi_{CL} \vdash \neg D(B) \Rightarrow \neg D(B)\, U_A\, D(I) \tag{CL2}$$

$$\phi_{CL} \vdash D(I) \twoheadrightarrow D(B) \tag{CL3}$$

$$\phi_{CL} \vdash D(I) \Rightarrow D(I)\, U_A\, D(B) \tag{CL4}$$

$$\phi_{CL} \vdash D(B) \Rightarrow D(B)\, U_A\, \neg D(I) \tag{CL5}$$

$$\phi_{CL} \vdash d(B) \Rightarrow d(B)\, U_A\, \neg d(I) \tag{CL6}$$

$$\phi_{CL} \vdash \neg d(I) \twoheadrightarrow \neg d(B) \tag{CL7}$$

$$\phi_{CL} \vdash \neg d(I) \Rightarrow \neg d(I)\, U_A\, \neg d(B) \tag{CL8}$$

where $\phi_{CL} \equiv_{def} \neg d(I)\ \&\ \neg d(B)$, the initial CL conditions. Note that speed-independence is once again assumed.

The align module synchronizes the PLs of the pipeline. It waits until all inputs are defined before it allows its output to be defined. When all inputs are defined, then it simultaneously defines the outputs (and so the align module is not internally speed-independent). We define the set of input and output lines for the align module are $I_a = A_C \cup B$ and $O_a = A_I \cup O$, respectively. Given this, we have align axioms

$$\phi_{align} \vdash \neg d(O_a) \Rightarrow \neg d(O_a)\, U_A\, D(I_a) \tag{A1}$$

$$\phi_{align} \vdash \neg d(I_a) \twoheadrightarrow \neg d(O_a) \tag{A2}$$

$$\phi_{align} \vdash D(O_a) \Rightarrow D(O_a)\, U_\Lambda \neg d(I_a) \tag{A3}$$

$$\phi_{align} \vdash D(I_a) \twoheadrightarrow D(O_a) \tag{A4}$$

$$\phi_{align} \vdash \Box(d(A_I) \equiv D(O)) \tag{A5}$$

where $\phi_{align} \equiv_{def} \neg d(I_a) \,\&\, \neg d(O_a)$. A1 and A3 state that no output line of the align module become defined until all of the inputs are defined. A2 and A4 are liveness axioms stating that when all of the inputs *are* defined that all of the outputs will eventually be defined. A5 states that when the outputs become either defined or undefined, they all do so simultaneously. Thus the output has two states, "all defined" or "all undefined."

The converter module, which converts a defined signal into an undefined one and vice-versa, has axioms

$$\phi_C \vdash \neg d(A_C) \Rightarrow \neg d(A_C)\, U_\Lambda \neg d(A_O) \tag{C1}$$

$$\phi_C \vdash \neg d(A_O) \twoheadrightarrow \neg d(A_C) \tag{C2}$$

$$\phi_C \vdash d(A_C) \Rightarrow d(A_C)\, U_\Lambda\, d(A_O) \tag{C3}$$

$$\phi_C \vdash d(A_O) \twoheadrightarrow \neg d(A_C). \tag{C4}$$

where $\phi_C \equiv_{def} d(A_O) \,\&\, \neg d(A_C)$.

Malachi and Owicki's specifications for the PL are

$$\phi_{PL} \vdash D(I) \Rightarrow D(I)\, U_\Lambda\, d(A_I) \tag{PL1}$$

$$\phi_{PL} \vdash \neg d(A_I) \Rightarrow \neg d(A_I)\, U_\Lambda \neg d(A_O) \,\&\, D(I) \tag{PL2}$$

$$\phi_{PL} \vdash \neg d(I) \Rightarrow \neg d(I)\, U_A\, \neg d(A_I) \quad \text{(PL3)}$$

$$\phi_{PL} \vdash d(A_I) \Rightarrow d(A_I)\, U_A\, \neg d(I)\ \&\ d(A_O) \quad \text{(PL4)}$$

$$\phi_{PL} \vdash D(O) \Rightarrow D(O)\, U_A\, d(A_O) \quad \text{(PL5)}$$

$$\phi_{PL} \vdash \neg d(A_O) \Rightarrow \neg d(A_O)\, U_A\, D(O) \quad \text{(PL6)}$$

$$\phi_{PL} \vdash \neg d(O) \Rightarrow \neg d(O)\, U_A\, \neg d(A_O)\ \&\ D(I) \quad \text{(PL7)}$$

$$\phi_{PL} \vdash d(A_O) \Rightarrow d(A_O)\, U_A\, \neg D(O) \quad \text{(PL8)}$$

$$\phi_{PL} \vdash D(I) \twoheadrightarrow d(A_I) \quad \text{(PL9)}$$

$$\phi_{PL} \vdash D(I) \twoheadrightarrow D(O) \quad \text{(PL10)}$$

$$\phi_{PL} \vdash D(O) \twoheadrightarrow d(A_O) \quad \text{(PL11)}$$

$$\phi_{PL} \vdash \neg d(I) \twoheadrightarrow \neg d(A_I) \quad \text{(PL12)}$$

$$\phi_{PL} \vdash \neg d(I) \twoheadrightarrow \neg d(O) \quad \text{(PL13)}$$

$$\phi_{PL} \vdash \neg d(O) \twoheadrightarrow \neg d(A_O) \quad \text{(PL14)}$$

where $\phi_{PL} \equiv_{def} \neg d(I)\ \&\ \neg d(B)\ \&\ d(A_O)\ \&\ \neg d(A_C)\ \&\ \neg d(O_a)$, the initial PL condition.

6.4.3 State Template and SLEG

For PL, a state template and corresponding EG with 18 states can be defined. Using the template of Table 6.1, it is possible to cover the execution. The state names have an "H" or "L" prefixing the set of lines which transitioned high (defined) or low

(undefined) in going from state to state.

The SLEG-formulas appearing in Table 6.2 were proved with the aid of the PTL decision procedure. For example, consider the state HA_O. When in this state, the following can be assumed.

(i) $d(I)$ is true.

(ii) $(d(B)\ U_A \neg d(I))$ by $d(B)$ and CL6.

(iii) $(d(A_O)\ U_A \neg d(O))$ by PL4 for right neighbor.

(iv) $d(A_C)$ is true.

(v) $D(O_a)\ U_A\ (\neg d(B)\ \&\ \neg d(A_C))$ by A3.

(vi) $(D(O_a) \Rightarrow D(O))$ by definition of O_a.

(vii) The liveness property that $d(A_O) \twoheadrightarrow \neg d(A_C)$ which allows the arcs out of HA_O to be tranformed to live arcs, using the Arc Transformation Theorem. The PTL timing diagram for this state and SLEG-formula appears in Figure 6.9. Using the DP with assumptions stated above, we deduce the SLEG-formula

$$HO_a \Rightarrow HO_a\ U^E\ (LA_C \vee LILA_C \vee LIHA_O)$$

It was necessary to assume that the left and right neighbor of PL obeyed the PL specifications, in order for the SLEG-formulas to be proved. The SLEG appears in Figure 6.10.

6.4.4 Checking PL Specifications

Let the initial state be LO_a. With the SLEG having been constructed, We prove the correctness of PL with respect to the Malachi/Owicki specifications using the SLEG. Since the specifications use U_A and not U, we first convert all PL specifications of the form $(u\ U_A\ v)$ into $(u\ U\ (u\ \&\ v))$. PL1 through PL14 can be

PL State Template	
State	Formula
LO_a	$\neg D(I)\ \&\ \neg D(B)\ \&\ d(A_O)\ \&\ \neg d(A_C)\ \&\ \neg d(O_a)$
HI	$D(I)\ \&\ \neg D(B)\ \&\ d(A_O)\ \&\ \neg d(A_C)\ \&\ \neg d(O_a)$
LA_O	$\neg D(I)\ \&\ \neg D(B)\ \&\ \neg d(A_O)\ \&\ \neg d(A_C)\ \&\ \neg d(O_a)$
HB	$D(I)\ \&\ D(B)\ \&\ d(A_O)\ \&\ \neg d(A_C)\ \&\ \neg d(O_a)$
$HILA_O$	$D(I)\ \&\ \neg D(B)\ \&\ \neg d(A_O)\ \&\ \neg d(A_C)\ \&\ \neg d(O_a)$
HA_C	$\neg D(I)\ \&\ \neg D(B)\ \&\ \neg d(A_O)\ \&\ d(A_C)\ \&\ \neg d(O_a)$
$HBLA_O$	$D(I)\ \&\ D(B)\ \&\ \neg d(A_O)\ \&\ \neg d(A_C)\ \&\ \neg d(O_a)$
$HIHA_C$	$D(I)\ \&\ \neg D(B)\ \&\ \neg d(A_O)\ \&\ d(A_C)\ \&\ \neg d(O_a)$
HA_CHB	$D(I)\ \&\ D(B)\ \&\ \neg d(A_O)\ \&\ d(A_C)\ \&\ \neg d(O_a)$
HO_a	$d(I)\ \&\ d(B)\ \&\ \neg d(A_O)\ \&\ d(A_C)\ \&\ D(O_a)$
LI	$\neg d(I)\ \&\ d(B)\ \&\ \neg d(A_O)\ \&\ d(A_C)\ \&\ D(O_a)$
HA_O	$d(I)\ \&\ d(B)\ \&\ d(A_O)\ \&\ d(A_C)\ \&\ D(O_a)$
LB	$\neg d(I)\ \&\ \neg d(B)\ \&\ \neg d(A_O)\ \&\ d(A_C)\ \&\ D(O_a)$
$LIHA_O$	$\neg d(I)\ \&\ d(B)\ \&\ d(A_O)\ \&\ d(A_C)\ \&\ D(O_a)$
LA_C	$d(I)\ \&\ d(B)\ \&\ d(A_O)\ \&\ \neg d(A_C)\ \&\ D(O_a)$
$LBHA_O$	$\neg d(I)\ \&\ \neg d(B)\ \&\ d(A_O)\ \&\ d(A_C)\ \&\ D(O_a)$
$LILA_C$	$\neg d(I)\ \&\ d(B)\ \&\ d(A_O)\ \&\ \neg d(A_C)\ \&\ D(O_a)$
LA_CLB	$\neg d(I)\ \&\ \neg d(B)\ \&\ d(A_O)\ \&\ \neg d(A_C)\ \&\ D(O_a)$

Table 6.1.

PL SLEG-Formulas	
State/Vertex	SLEG-Formula
LO_a	$LO_a \Rightarrow LO_a\ U^E\ (LA_O \vee HILA_O \vee HI)$
HI	$HI \Rightarrow HI\ U^E\ (HILA_O \vee HBLA_O \vee HB)$
LA_O	$LA_O \Rightarrow LA_O\ U^E\ (HA_C \vee HIHA_C \vee HILA_O)$
HB	$HB \Rightarrow HB\ U^E\ HBLA_O$
$HILA_O$	$HILA_O \Rightarrow HILA_O\ U^E\ (HIHA_C \vee HA_C HB \vee HBLA_O)$
HA_C	$HA_C \Rightarrow HA_C\ U\ HIHA_C$
$HBLA_O$	$HBLA_O \Rightarrow HBLA_O\ U^E\ HA_C HB$
$HIHA_C$	$HIHA_C \Rightarrow HIHA_C\ U^E\ HA_C HB$
$HA_C HB$	$HA_C HB \Rightarrow HA_C HB\ U^E\ HO_a$
HO_a	$HO_a \Rightarrow HO_a\ U^E\ (HA_O \vee LIHA_O \vee LI)$
LI	$LI \Rightarrow LI\ U^E\ (LIHA_O \vee LBHA_O \vee LB)$
HA_O	$HA_O \Rightarrow HA_O\ U^E\ (LA_C \vee LILA_C \vee LIHA_O)$
LB	$LB \Rightarrow LB\ U^E\ LBHA_O$
$LIHA_O$	$LIHA_O \Rightarrow LIHA_O\ U^E\ (LILA_C \vee LA_C LB \vee LBHA_O)$
LA_C	$LA_C \Rightarrow LA_C\ U\ LILA_C$
$LBHA_O$	$LBHA_O \Rightarrow LBHA_O\ U^E\ LA_C LB$
$LILA_C$	$LILA_C \Rightarrow LILA_C\ U^E\ LA_C LB$
$LA_C LB$	$LA_C LB \Rightarrow LA_C LB\ U^E\ LO_a$

Table 6.2.

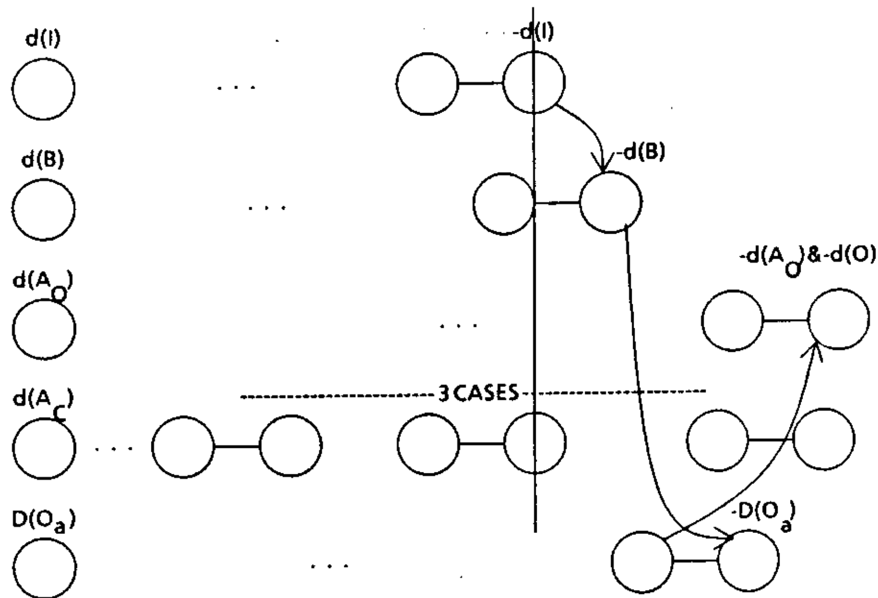


Figure 6.9. PTL Timing Diagram for PL State

then be checked easily by inspection of the SLEG, starting in the states where ξ is true and following all paths until condition is met. The formal justification is provided by the Safety Checking and Liveness Checking Theorems.

This self-timed pipeline example demonstrates that with the execution graph approach, we do not have to specify the value of every line variable to be able to construct a state graph. Reasoning is done at the module level; it is not necessary to go down to the gate level. The EG and SLEG need only be detailed enough to be able to show the desired correctness assertions.

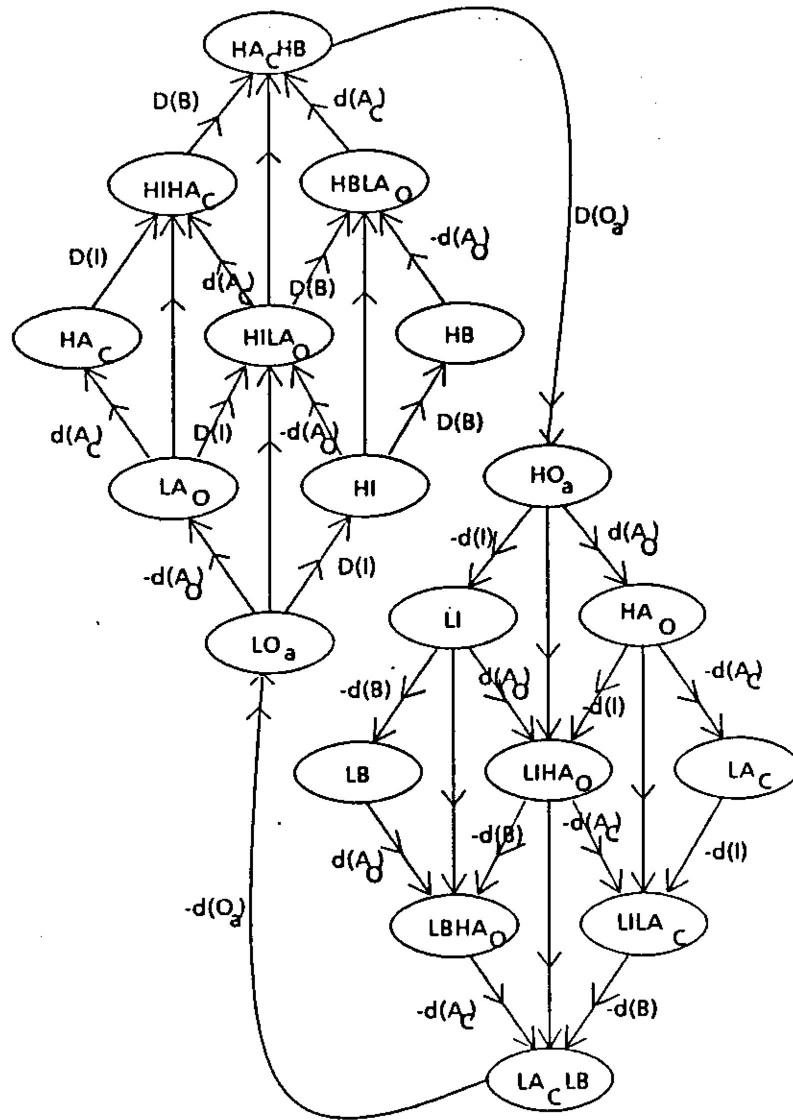


Figure 6.10. SLEG for PL

CHAPTER 7

Anomalies

Thus far we have only considered proving desirable properties such as liveness and safety. For the controller in Chapter 4, the liveness property proved was that the correct state change eventually occurs when a correct input is applied. The safety property proved was that it remains stable in the correct state after the transition. We have not yet, however, specifically considered incorrect operation. We did not show, for example, that the controller does not go through any erroneous states during the transition to the correct state. This property would be the absence of a race.

Our focus now turns to properties involving anomalies. As a starting point, it is first shown how temporal logic can be used to reason about the presence of a race in a classical adder. We then show how to reason about the time behavior of inverter chains Oscillation appears once again in this context. Finally we prove that the asynchronous controller does not enter erroneous states.

7.1 Proving the Presence of Critical Races

A race is present in an asynchronous sequential circuit when two or more state

variables are changing simultaneously, causing the circuit to go through an indeterminate sequence of intermediate (non-final) states. A critical race is a race which causes the circuit not to settle or to reach an incorrect state and settle there for a long enough period of time to be considered a problem. Before proving race properties, let us extend PTL to deal with spurious oscillation.

7.1.1 Grammar Operators

Wolper introduces the idea of augmenting temporal logic with additional operators as needed [Wol83]. Showing that temporal logic is not as expressive as one might think, he motivates the idea of writing right-linear grammar rules which describe state sequences satisfying new operators. He gives a general axiomatization for any temporal logic with $\bigcirc$ and any set of operators defined by right-linear rules. He also shows how the standard operators may be regarded as grammar operators. For example, the standard condition $p\ U\ q$ could be described by the right-linear grammar

$$V_0 \rightarrow p\ V_0$$

$$V_0 \rightarrow q.$$

Temporal logic does not have an operator for the infinite alternating sequence

$$p\ q\ p\ q\ \cdots.$$

Using Wolper's method, we introduce a pair of *oscillation operators,* Ω_0 and Ω_1, where $p\ \Omega_0\ q$ means "p now, then q, then p, then q, ..." and $p\ \Omega_1\ q$ means "q now, then p, then q, then p, ...". The grammar rules are

$$V_0 \rightarrow pV_1$$

$$V_1 \rightarrow qV_0.$$

V_0 generates unbounded words $pqpq \cdots$ on this grammar. A complete axiomatic system for temporal logic with oscillation operators has oscillation axioms:

$$\vdash\ p\ \Omega_0\, q\ \supset\ p\ \&\ \bigcirc(p\ \Omega_1\, q) \tag{O1}$$

$$\vdash\ u_0\ \&\ \square(u_0 \supset p\ \&\ \bigcirc u_1)\ \&\ \square(u_1 \supset q\ \&\ \bigcirc u_0)\ \supset\ p\ \Omega_0\, q \tag{O2}$$

$$\vdash\ p\ \Omega_1\, q\ \supset\ q\ \&\ \bigcirc(p\ \Omega_0\, q) \tag{O3}$$

$$\vdash\ u_1\ \&\ \square(u_0 \supset p\ \&\ \bigcirc u_1)\ \&\ \square(u_1 \supset q\ \&\ \bigcirc u_0)\ \supset\ p\ \Omega_1\, q \tag{O4}$$

We require that $\bigcirc$ and Ω_0 by related by

$$\models \bigcirc(u\ \Omega_0\, v)\ \equiv\ (\bigcirc u)\,\Omega_0\,(\bigcirc v) \tag{T35}$$

This is shown valid by the semantics of $\bigcirc$ and Ω_0.

Proof

$$\begin{aligned}
\mu[\![\bigcirc(u\ \Omega_0\, v)]\!](\bar{\sigma}) \ &\equiv\ \mu[\![u\ \Omega_0\, v]\!](\bar{\sigma}^{(1)}) \\
&\equiv\ \forall i{\ge}0.[\ even(i) \supset \mu[\![u]\!]((\bar{\sigma}^{(1)})^{(i)})\ \&\ odd(i) \supset \mu[\![v]\!]((\bar{\sigma}^{(1)})^{(i)})\ ] \\
&\equiv\ \forall i{\ge}0.[\ even(i) \supset \mu[\![u]\!](\bar{\sigma}^{(i+1)})\ \&\ odd(i) \supset \mu[\![v]\!]((\bar{\sigma}^{(i+1)})\ ] \\
&\equiv\ \forall i{\ge}0.[\ even(i) \supset \mu[\![\bigcirc u]\!](\bar{\sigma}^{(i)})\ \&\ odd(i) \supset \mu[\![\bigcirc v]\!](\bar{\sigma}^{(i))}\ ] \\
&\equiv\ \mu[\![(\bigcirc u)\,\Omega_0\,(\bigcirc v)]\!](\bar{\sigma})
\end{aligned}$$

7.1.2 Reasoning About Races

The classical end-around-carry, one's-complement adder of Figure 7.1 has a critical race which theoretically can manifest itself in infinite oscillation for certain inputs and certain gate delays. In practice, the infinite oscillation eventually settles in an indeterminate state. However, it may take so long to do so that a correct result is not available when needed. Shedletsky points out this problem [She77].

We may regard the adder with the end-around-carry as an asynchronous sequential circuit (sequential because of the end-around carry feedback). For the inputs shown in Figure 7.1(b) the adder produces a correct output, leaving the carry lines at 0101. These left-over carry values cause the oscillation problem for the subsequent addition of 1101 and 0010 (Figure 7.1(c)) if the delays of the carries are equal. The situation can be modelled in our extended temporal logic with Ω_0 and Ω_1 .

If we replace our unbounded delay gate axioms with unit delay axioms, the problem will manifest itself. Let the adder majority and exclusive-or functions be defined as follows.

$$Maji \equiv_{def} (x_i \ \& \ y_i) \vee (x_i \ \& \ c_i) \vee (y_i \ \& \ c_i)$$

$$Xori \equiv_{def} x_i \ XOR \ y_i \ XOR \ z_i$$

Unit delay adder axioms are (regarding the adders as the smallest device to be considered)

$$Maji \Rightarrow \bigcirc c_{i+1} \qquad \text{(Carry1)}$$

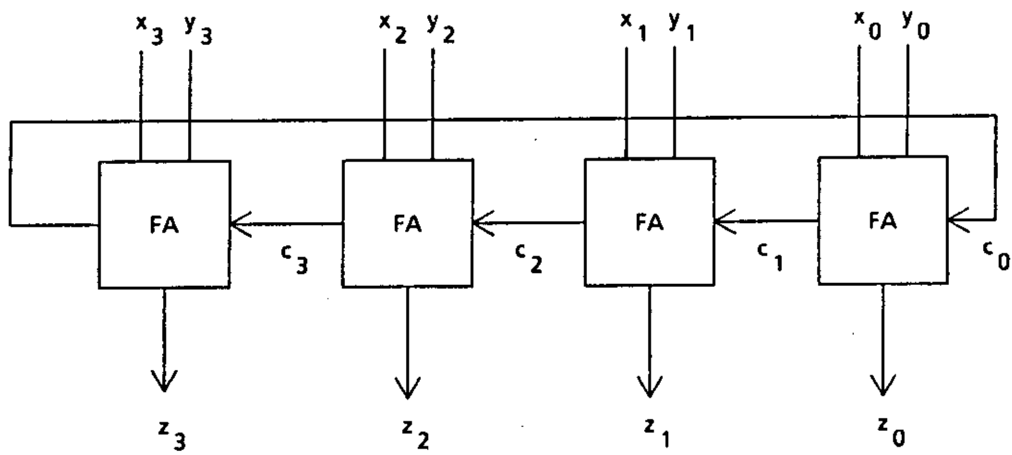


Figure 7.1 End-Around-Carry Adder

$$\neg Maji \Rightarrow \bigcirc \neg c_{i+1} \tag{Carry2}$$

$$c_{i+1} \Rightarrow c_{i+1}\ U_A\ \neg Maji \tag{Carry3}$$

$$\neg c_{i+1} \Rightarrow \neg c_{i+1}\ U_A\ Maji \tag{Carry4}$$

$$Xori \Rightarrow \bigcirc z_i \tag{Sum1}$$

$$\neg Xori \Rightarrow \bigcirc \neg z_i \tag{Sum2}$$

$$z_i \Rightarrow z_i\ U_A\ \neg Xori \tag{Sum3}$$

$$\neg z_i \Rightarrow \neg z_i\ U_A\ Xori \tag{Sum4}$$

where the c_i are carries and the z_i are outputs. Let the assertion that the set of lines $A = \{a_{n-1}, \cdots, a_0\}$ have Boolean values $i_{n-1}, \cdots, i_0$ ($i_k = 0\ or\ 1$) be written $A_{i_{n-1}, \cdots, i_0}$ (e.g. $c_{1010} \equiv c_3\ \&\ \neg c_2\ \&\ c_1\ \&\ \neg c_0$). Define

$$CarriesSettled \equiv_{def} c_{0000},$$

the initial carry line values. The inputs x and y are assumed to be 1010 and 1010 for 1 time-unit, then 1101 and 0010 forever after that. This is the simplest input which can be applied for the oscillation problem to arise. The inputs 1010 and 1010 set the carry lines to 0101, which will then oscillate between 0101 and 1010 while the second input is applied. The input assumption is

$$A_0 \equiv_{def} CarriesSettled\ \&\ (x_{1010}\ \&\ y_{1010})\ \&\ \bigcirc \Box (x_{1101}\ \&\ y_{0010})$$

We prove three conditions in the antecedent of O2. The first is

$$\vdash\ A_0 \supset \bigcirc c_{0101} \tag{7.1}$$

This is proved by PC. Next we show, for any n,

$$\vdash A_0 \supset \bigcirc\bigcirc^n (x_{1101} \,\&\, y_{0010})$$

by the PTL axiom A5 $\vdash \Box w \supset (w \,\&\, \bigcirc w \,\&\, \bigcirc \Box w)$ (use n times to get $\Box w \supset \bigcirc^n w$, then use axiom A3 $\vdash \bigcirc(w_1 \supset w_2) \supset (\bigcirc w_1 \supset \bigcirc w_2)$.) Also, from the adder axioms,

$$\vdash \bigcirc\bigcirc^n(x_{1101} \,\&\, y_{0010}) \,\&\, \bigcirc\bigcirc^n c_{0101} \supset \bigcirc\bigcirc^{n+1}(c_{1010} \,\&\, z_{1010})$$

So

$$\forall n \geq 0.(\vdash A_0 \supset \bigcirc\bigcirc^n [c_{0101} \supset \bigcirc(c_{1010} \,\&\, z_{1010})]) \tag{7.2}$$

We now need the validity of a rule we will call *generalization of deduction,* stated

$$\frac{(\vdash P(0)) \,\&\, \cdots \,\&\, (\vdash P(k))}{\vdash (P(0) \,\&\, \cdots \,\&\, P(k))} \tag{GD}$$

for *all* $k \geq 0$.

Proof The proof is by induction on k,

basis $k = 0$. Trivially $\vdash P(0)$ implies $\vdash P(0)$,

induction Assume the theorem for k. Proving the theorem for $k+1$: if

$$\vdash P(0) \,\&\, \cdots \,\&\, \vdash P(k) \,\&\, \vdash P(k+1)$$

then, by the induction hypothesis for k,

$$\vdash \forall i, 0 \leq i \leq k.\ P(k) \,\&\, \vdash P(k+1).$$

This is

$$\vdash F(k)\ \&\ \vdash P(k+1)$$

to which we apply the induction hypothesis for index 2, yielding

$$\vdash (F(k)\ \&\ P(k+1)).$$

Exapnding F,

$$\vdash \forall i, 0 \le i \le k.\ P(i)\ \ \&\ \ P(k+1).$$

We recognize this as

$$\vdash \forall i, 0 \le i \le k+1.\ P(i)$$

•

Applying GD to (7.2) and moving the "$\forall n \ge 0$" inside (since n is not free in A_0) yields

$$\vdash\ A_0 \supset \bigcirc \forall n \ge 0.\ \bigcirc^n [\, c_{0101} \supset \bigcirc(c_{1010}\ \&\ z_{1010})\,]$$

Since $\Box w \equiv_{def} \forall n \ge 0. \bigcirc^n w$, we have

$$\vdash\ A_0 \supset \bigcirc \Box [\, c_{0101} \supset \bigcirc(c_{1010}\ \&\ z_{1010})\,]. \qquad (7.3)$$

Similarly we get

$$\vdash\ A_0 \supset \bigcirc \Box [\, c_{1010} \supset \bigcirc(c_{0101}\ \&\ z_{0101})\,]. \qquad (7.4)$$

Combining (7.1), (7.3), and (7.4) we get:

$$\begin{aligned} \vdash\ A_0\ \supset\ \bigcirc[\, c_{0101}\ &\&\ \Box(c_{0101} \supset \bigcirc(c_{1010}\ \&\ z_{1010})) \qquad (7.5) \\ &\&\ \Box(c_{1010} \supset \bigcirc(c_{0101}\ \&\ z_{0101}))\,] \end{aligned}$$

Formula (7.5) is in the form for axiom O2. Applying it with

$$u_0 \equiv c_{0101}$$

$$u_1 \equiv c_{1010}$$

$$p \equiv \bigcirc z_{1010}$$

$$q \equiv \bigcirc z_{0101},$$

leaves us with

$$\vdash A_0 \supset \bigcirc((\bigcirc z_{0101})\, \Omega_0\, (\bigcirc z_{1010}))$$

Applying T35 yields

$$\vdash A_0 \supset \bigcirc^2(z_{0101}\, \Omega_0\, z_{1010})$$

This last formula says, based on our assumptions A_0, that the adder will oscillate forever.

7.2 Stability and Instability

As a demonstration of the use of forward and backward reasoning in the context of stability, we consider the time behavior of closed inverter chain loops. We know from experience that a loop consisting of an odd number of inverters will oscillate forever, never stabilizing, and that an even number of inverters will eventually settle in a stable state, no matter what the initial state is.

The inverter chain loop of Figure 7.2 has n inverters. If n is odd then, for initial values 1010..., we have the following gate axioms.

$$\vdash L_1 \twoheadrightarrow \neg L_2$$

$$\vdash \neg L_2 \twoheadrightarrow L_3$$

$$\ldots$$

$$\vdash \neg L_{n-1} \twoheadrightarrow L_n$$

$$\vdash L_n \twoheadrightarrow \neg L_1$$

Using FR $n-1$ times gives us

$$\vdash L_1 \twoheadrightarrow \neg L_1$$

Continuing the same reasoning, we have

$$\vdash \neg L_1 \twoheadrightarrow L_2$$

$$\vdash L_2 \twoheadrightarrow \neg L_3$$

$$\ldots$$

$$\vdash L_{n-1} \twoheadrightarrow \neg L_n$$

$$\vdash \neg L_n \twoheadrightarrow L_1$$

Using FR $n-1$ times gives us

$$\vdash \neg L_1 \twoheadrightarrow L_1$$

So

$$\vdash \Box(L_1 \supset \nabla \neg L_1) \;\&\; \Box(\neg L_1 \supset \nabla L_1)$$

and, by T4,

$$\vdash \Box((L_1 \supset \nabla \neg L_1) \,\&\, (\neg L_1 \supset \nabla L_1)),$$

stating that whenever L_1 goes high it will eventually go low, and whenever it goes low it will go high, never settling. In fact, using the same reasoning, we can conclude

$$\vdash \Box((L_i \supset \nabla \neg L_i) \,\&\, (\neg L_i \supset \nabla L_i))$$

for all i.

If n is even, then we have the axioms

$$\vdash L_1 \twoheadrightarrow \neg L_2$$

$$\vdash \neg L_2 \twoheadrightarrow L_3$$

$$\cdots$$

$$\vdash L_{n-1} \twoheadrightarrow \neg L_n$$

$$\vdash \neg L_n \twoheadrightarrow L_1$$

which do not yield $L_i \supset \nabla \neg L_i$. So we cannot derive the instability property as before, which is to be expected, but, by using GBR, we can derive the stability. The axioms, assuming initial values of 1010..., are

$$\vdash L_1 \Rightarrow L_1 \, U_A \, L_2$$

$$\vdash \neg L_2 \Rightarrow \neg L_2 \, U_A \, \neg L_3$$

$$\cdots$$

$$\vdash L_{n-1} \Rightarrow L_{n-1}\ U_A\ L_n$$

$$\vdash \neg L_n \Rightarrow \neg L_n\ U_A\ \neg L_{n-1}.$$

Using GBR we derive

$$\vdash (L_1 \,\&\, \neg L_2 \,\&\, \cdots \,\&\, L_{n-1} \,\&\, \neg L_n) \Rightarrow$$

$$(L_1 \,\&\, \neg L_2 \,\&\, \cdots \,\&\, L_{n-1} \,\&\, \neg L_n)\ U_A\ \neg L_1$$

which can be reduced to

$$\vdash (L_1 \,\&\, \neg L_2 \,\&\, \cdots \,\&\, L_{n-1} \,\&\, \neg L_n) \Rightarrow$$

$$((L_1 \,\&\, \neg L_2 \,\&\, \cdots \,\&\, L_{n-1} \,\&\, \neg L_n)\ U_A\ \text{false})$$

by the definition of U_A and, finally,

$$(L_1 \,\&\, \neg L_2 \,\&\, \cdots \,\&\, L_{n-1} \,\&\, \neg L_n) \Rightarrow \qquad (7.6)$$

$$\Box (L_1 \,\&\, \neg L_2 \,\&\, \cdots \,\&\, L_{n-1} \,\&\, \neg L_n)$$

by the definition of U_A and the recusive definition of $\Box$, which is

$$\Box w \equiv_{def} w \,\&\, \bigcirc \Box w.$$

Formula 7.6 states that the situation 1010... for even inverter loops is stable.

7.3 Proving the Absence of Critical Races

Backward reasoning can be used to prove the absence of a critical race. For the controller example, a race might cause the controller to be in a state other that q_i or q_{i+1} for some period of time. This could certainly be a problem. Define

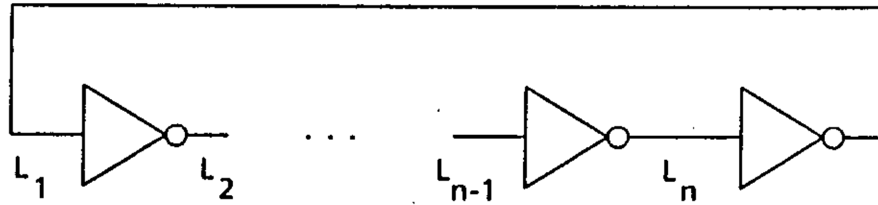


Figure 7.2 Inverter Chain Loop

$$WrongState \equiv_{def} IS(q_{i+2}) \vee \cdots \vee IS(q_{i-1}),$$

so that we are in a bad state for the transition $q_i \rightarrow q_{i+1}$ if we go through any of q_{i+2}, ... , q_{i-1}. Each cell has a safety property $(Reset_j\ U_A\ IS(q_{i+1}))$, for $j = i+2, \cdots, i-1$.

$\vdash A_{SC} \supset (\neg Q_j \ \&\ \overline{Q}_j)\ U_A \neg \overline{S}_j$ by $LatchStayReset_j$

$\vdash A_{SC} \supset \overline{S}_j\ U_A\ (E_j \ \&\ P_j)$ by gate axiom

$\vdash A_{SC} \supset \neg P_j\ U\ IS(q_{i+1})$ by def. if A_{SC}

$\vdash A_{SC} \supset \neg Q_j \ \&\ \overline{Q}_j\ U_A\ IS(q_{i+1})$ by BR twice

We also know since

$$\vdash IS(q_j) \supset Q_j \mathbin{\&} \neg \overline{Q}_j$$

that the contrapositive

$$\vdash \Box(\neg Q_j \vee \overline{Q}_j \supset \neg IS(q_j)) \tag{7.7}$$

is true (introducing $\Box$ by $\Box$I.) The last formula of our deduction above becomes

$$\vdash A_{SC} \supset (\neg Q_j \vee \overline{Q}_j)\, U_A\, IS(q_{i+1})$$

by T25. This yields

$$\vdash \neg IS(q_j)\, U_A\, IS(q_{i+1}) \tag{7.8}$$

by T25 again using formula 7.7.

Combining each of the $n-2$ properties of the form (7.8) using T6 finally yields

$$\vdash A_{SC} \supset (\neg \mathit{WrongState}\ U_A\ IS(q_{i+1})). \tag{7.9}$$

Formula 7.9 states that the controller will not enter a wrong state during the state change $q_i \rightarrow q_{i+1}$.

CHAPTER 8

Conclusions and Suggestions for Future Research

8.1 Summary

The goal of this research was to provide a framework for proving the correctness of speed-independent asynchronous circuits. Speed-independent circuits operate correctly regardless of the distribution of gate delays. Temporal logic is a natural vehicle since in temporal logic it is very natural and common to say that an event will "eventually" occur without specifying exactly when. We focused on various speed-independent circuits, specifying and proving their safety and liveness properties.

We have found the standard linear-time PTL (propositional TL) with the four operators $\Box$, ∇, $\bigcirc$, and U and no quantification to be quite appropriate for our application. This is also the version which has been most widely accepted by the computer science research community. We did, however, add some operators. The until-after (U_A) operator is introduced for the axiomatization of gate behavior because we need a form of U which guarantees that at some point in time the input and output of a gate will be such that the gate is not excited. $\twoheadrightarrow$ and $\Rightarrow$ are added merely for convenience. We also found that Wolper's ETL (Extended TL) [Wol83] is quite useful for defining operators which describe event sequences not expressible in PTL. In

particular, oscillation is handled easily by introducing an oscillation operator and reasoning in ETL.

Assuming that gates have finite, unbounded delay and that lines have no delay, we proceeded to axiomatize gate behavior. We also discussed the alternative assumption of unit-delay.

Applying input forever allowed us to focus on steady-state properties, in the meantime establishing the flavor of the TL deduction process. If one were to apply an input to a circuit forever and observe the corresponding response, the circuit would settle into some steady-state. This liveness is expressed

$$IC \ \& \ \Box Stimulus \supset \nabla \Box Response$$

where IC describes the initial conditions (initial circuit state), *Stimulus* describes some input stimulus to the circuit, and *Response* describes some state the circuit will remain in until further stimuli. We established some proof rules (theorems of PTL) which allowed us to construct proofs of safety and liveness properties. The first example circuit was a latch composed of two cross-coupled NAND gates. Then using the latch and the gate again as primitive elements, an asychronous controller composed of these devices was proved correct.

In the course of the controller proof we proved non-interference properties which state that certain portions of the circuit will not affect the behavior of another portion of interest. The notion of non-interference is similar to that of concurrent programs with shared memory. In the case of asynchronous circuits the analog of the shared variables are "shared" lines.

If the steady-state frame of time can be regarded as "local-time" then we are certainly interested in properties which are valid if the input is *not* applied forever. These

properties are valid for all future time. Global-time time properties are safety or liveness properties which are valid throughout out the time of normal circuit operation. Global-time liveness properties take the form

$$IC \ \& \ (Stimulus \ U_A \ LongEnough) \supset \nabla (Response \ U_A \ NextStimulus)$$

where IC is a formula describing the initial condition (initial state), *LongEnough* is a formula which is true when the stimulus has been sufficiently recognized by the circuit, and *NextStimulus* is true when nothing more can be assumed regarding the stability of the circuit's response.

The method of proving global-time properties is a heuristic application of techniques such as using forward and backward reasoning, using global-time versions of the proof rules introduced in the discussion on steady-state properties, and using known PTL theorems for reasoning. Forward and backward reasoning could be regarded as proof rules also.

Programs have an execution which can be inferred from the semantics of the language in which they are written. One of the problems with verifying a circuit given its schematic diagram is that no representation of its execution is available. One may construct a finite state graph to represent the execution and then reason about the graph, but there is no guarantee the graph, and therefore the reasoning, is correct. Our Execution Graph Method (EGM) provides a formal justification for the contruction of a circuit execution graph. The graph can used as a tool for reasoning about safety and liveness properties. An application of EGM to proving global-time properties of self-timed systems was considered; an asynchronous pipeline was verified, and in doing this, we tested the method on a state-of-the-art VLSI device.

We also used temporal logic to reason about the presence and absence of races.

(i) Using Wolper's ETL and an oscillation operator which we created (Ω_0), we proved that an end-around carry adder will oscillate forever for some distribution of delays and some inputs (presence of a race).

(ii) Using forward reasoning, we proved that a loop consisting of an odd number of inverters will oscillate forever.

(iii) Using backward reasoning, we proved that a loop consisting of an even number of inverters which are initially unexcited will be stable forever.

(iv) Using our techniques for global-time properties, we proved that an example asynchronous controller never enters a bad state (absence of a races).

8.2 Relation to Other Work

In philosophy, our approach is consistent with the idea that one constructs a logic in order to reason abstractly about variables and their values (states). Other approaches ([Cla83, Mis83, Dil85]) involve reasoning exclusively about states rather than higher-level specifications.

The unbounded delay assumption means we must only work with speed-independent circuits. Within this class, however, all verification is quite practical. We did not assume some simpler model such as unit-delay and prove properties which rely on this restrictive assumption.

Our approach has the desirable "functional specifications" feature: specifications of higher level modules are functions of the specifications of lower level modules.

We are not limited to finite state machine underlying models. Thus reasoning about groups of lines and devices as non-Boolean-valued objects is possible. This would allow reasoning about data as well as control. Our approach requires human

interaction, possibly aided by próof checkers and theorem provers. This relatively manual approach has the advantage that the person doing the proof is aware of what steps must take place and, in fact, directs most of them. If the proof is not successful, the person has a instinct for why not. He may need to change the original specification of the desired property. This arose in the global-time liveness proof for the controller.

Since PTL is decidable, one may raise the suggestion that instead of proving everything within the PTL deductive system, why not just axiomatize the gates or modules and feed the circuit specification and the axioms into the DP? There are three reasons why this might not be a good solution. First, we cannot assume that PTL will be powerful enough to specify the behavior of all circuits of interest. As soon as we abstract a little and treat groups of Boolean lines as integers, we go outside the expressibility and decidability of PTL. We need to use a quantified temporal logic which consists of PTL augmented with quantifiers, data types, functions, and predicates. A quantified temporal logic would be undecidable, therefore the DP would only be of value on the decidable subset. Relying upon the decidability of the specification language is limiting. Performing proofs in the deductive system, as we have done, does not depend upon PTL's decidability. A quantified temporal logic could be treated with our methods. Secondly, for the reasons discussed above we want control over the proof process. A completely automatic method is fine if the assertion turns out to be provable. If the assertion is not provable, however, the automatic method helps very little identifying what went wrong. Thirdly, if the DP were given full control, we would be placing all trust in the DP, a program which is not guaranteed correct itself. Without seeing the safety/liveness execution graph (SLEG), the user has an invisible proof with only the "yes" or "no" answer for evidence.

With our approach intermediate theorems and the SLEG provide an intuitive explanation of why the circuit is believed to be correct.

8.3 Future Directions

EGM featured a method for checking safety and liveness formulas on a safety/liveness execution graph. Any specification in one of two forms $\xi \Rightarrow \xi \, U_A \, \eta$ or $\xi \twoheadrightarrow \eta$ may be checked using the Safety Checking and Liveness Checking Theorems. A logical extension of EGM would allow nested specification w where w is any PTL formula. The algorithms for safety and liveness would require greater sophisication. Recursive traversal of the SLEG would be required.

Automating the process is necessary in order to deal with the complexities of realistic circuits. Automation comes in two forms: proof checking and proof contruction. Fully-automatic proof checkers could be written which would know the PTL axioms, PTL inference rules, gate axioms, proof rules, and PTL theorems. They would be used to check proofs.

Much of the proof contruction in EGM can be automated. After the human selects a state template, the SLEG-generation algorithm of Section 6.3 could be invoked to build the SLEG. Once the SLEG is built, the algorithms of the Safety Checking and Liveness Checking Theorems can be used to check safety and liveness assertions on the SLEG. For circuits which are built in a highly modular fashion, the safety and liveness checking could done by hand as it was for the asynchronous pipeline.

8.4 Conclusions

From a theoretical standpoint, we have been able to demonstrate that the correctness of circuits can be proven in a formal temporal logic deductive system. Throughout the presentation the deductive system was used directly or indirectly (indirectly in the case of EGM). The steady-state and global-time proof techniques were a purely constructive approach. It was seen in Chapters 4 and 5 that as circuits were constructed from devices, the specifications for the devices were manipulated into the specfication of the circuit through the use of PTL theorems and proof rules. The complexity of this process began to take its toll, however, and we introduced a method which included behavioral information, EGM.

EGM has the advantage that a collection of behavioral information is organized into a structure, a SLEG, which serves as a resource for checking specifications. We can conclude that circuit verification requires a behavioral description in order to be practical, even though it is possible to proceed without one (as was done in Chapters 4 and 5).

The concepts presented here are not limited to asynchronous circuits, but could be applied to any parallel asynchronous system of computing agents. The flow of signals between gates, for example, models the passing of messages between processes in distributed systems.

In the past, Boolean algebra has been the only mathematical tool for logic design. Temporal logic adds the ability to formally reason about time behavior at the logic level. Even if it turns out be impractical to verify every circuit, the formalization of hardware behavior leads to better understanding, better modelling, and, as a result, better designs.

APPENDIX A

Proofs of Theorems 4.1 - 4.7

Theorems 4.1 through 4.7 state the validity of the steady-state proof rules. We prove the theorems here.

Theorem 4.1. The implications

$$[\Box \neg \alpha_1] \ \& \ [\beta \Rightarrow \beta \, U_A \, (\alpha_1 \vee \alpha_2)] \ \supset \qquad \text{(IEOSimp1)}$$

$$[\beta \Rightarrow \beta \, U_A \, \alpha_2]$$

and

$$[\Box \alpha_1] \ \& \ [\beta \Rightarrow \beta \, U_A \, (\alpha_1 \vee \alpha_2)] \ \supset \qquad \text{(IEOSimp2)}$$

$$[\beta \Rightarrow \beta \, U_A \, \alpha_2].$$

are valid.

Proof The implication

$$\models [\Box \neg \alpha_1] \;\&\; [\beta \Rightarrow \beta\, U_A\, (\alpha_1 \vee \alpha_2)] \;\supset$$ (IEOSimp1)

$$[\beta \Rightarrow \beta\, U_A\, \alpha_2].$$

is true intuitively since, if β is waiting for either α_1 or α_2 and if α_1 will never become true, then β is effectively just waiting for α_2. First the validity of the implication

$$[\Box \neg \alpha_1] \;\&\; [\beta \supset \beta\, U_A\, (\alpha_1 \vee \alpha_2)] \;\supset\; [\beta \supset \beta\, U_A\, \alpha_2] \qquad \text{(A.1)}$$

will be shown in the semantics. This amounts to showing that μ of the implication is true for any state sequence $\bar{\sigma}$. Taking μ of the antecedent, we obtain $\mathbf{A}$.

$$\mathbf{A} \;\equiv\; \forall i{\ge}0.\mu[\![\neg \alpha_1]\!](\bar{\sigma}^{(i)}) \;\&\; (\neg\, \mu[\![\beta]\!](\bar{\sigma}) \vee \forall i{\ge}0.\mu[\![\beta]\!](\bar{\sigma}^{(i)})$$

$$\vee\, \exists i{\ge}0.\mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(i)}) \;\&\; \forall j,0{\le}j{\le}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)}))$$

Let us abbreviate

$$\psi_0 \equiv \forall i{\ge}0.\mu[\![\neg \alpha_1]\!](\bar{\sigma}^{(i)})$$

$$\psi_1 \equiv \neg\, \mu[\![\beta]\!](\bar{\sigma})$$

$$\psi_2 \equiv \forall i{\ge}0.\mu[\![\beta]\!](\bar{\sigma}^{(i)})$$

$$\psi_3 \equiv \exists i{\ge}0.\mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(i)}) \;\&\; \forall j,0{\le}j{\le}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)}).$$

Then $\mathbf{A} \equiv \psi_0 \;\&\; (\psi_1 \vee \psi_2 \vee \psi_3)$. Breaking up ψ_3 by applying μ to $\alpha_1 \vee \alpha_2$ yields

$$\psi_3 \;\equiv\; \exists i{\ge}0.[\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \vee \mu[\![\alpha_2]\!](\bar{\sigma}^{(i)}) \,] \;\&\; \forall j,0{\le}j{\le}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)}).$$

Distributing $\forall j,0{\le}j{\le}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)})$ and applying the predicate calculus theorem

$$\exists i.(P(i) \;\&\; Q(i)) \;\supset\; (\exists i.P(i)) \;\&\; (\exists i.Q(i)) \qquad \text{(PredC1)}$$

yields

$$A' \equiv \psi_0 \,\&\, (\psi_1 \vee \psi_2 \vee \psi_{\alpha_1} \vee \psi_{\alpha_2})$$

where

$$\psi_{\alpha_1} \equiv \exists i{\ge}0.\mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,\&\, \forall j, 0{\le}j{\le}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)})$$

and

$$\psi_{\alpha_2} \equiv \exists i{\ge}0.\mu[\![\alpha_2]\!](\bar{\sigma}^{(i)}) \,\&\, \forall j, 0{\le}j{\le}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)})$$

By the same theorem, PredC1,

$$\psi_{\alpha_1} \supset \exists i{\ge}0.[\,\mu[\![\alpha_1]\!](\bar{\sigma}^{(i)})\,] \,\&\, \exists i{\ge}0.[\,\forall j, 0{\le}j{\le}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)})].$$

But this conflicts with ψ_0 so ψ_{α_1} cannot be true. This leaves $A' \equiv \psi_0 \,\&\, (\psi_1 \vee \psi_2 \vee \psi_{\alpha_2})$ which implies $\psi_1 \vee \psi_2 \vee \psi_{\alpha_2}$ which is $\mu[\![\beta \supset \beta\, U_A\, \alpha_2]\!](\bar{\sigma})$. Hence the validity of formula A.1 is established. Implication IEOSimp1 follows by the $\Box\,\Box$ Rule, yielding

$$\vdash \Box([\Box\neg\alpha_1] \,\&\, [\beta \supset \beta\, U_A\, (\alpha_1 \vee \alpha_2)]) \supset \Box[\beta \supset \beta\, U\, \alpha_2],$$

and then distributing $\Box$ over $\&$ (by T4 and ER) and reducing $\Box\,\Box\alpha_1$ to $\Box\alpha_1$ (by T1 and ER).

The implication IEOSimp2 is proved similarly.

•

Theorem 4.2. The implications

$$\vdash [\Box \neg \alpha_1] \ \& \ [(\alpha_1 \vee \alpha_2) \Rightarrow (\alpha_1 \vee \alpha_2) \, U_A \, \beta] \ \supset \qquad \text{(IEISimp1)}$$

$$[(\alpha_1 \vee \alpha_2) \Rightarrow \alpha_2 \, U_A \, \beta]$$

and

$$\vdash [\Box \alpha_1] \ \& \ [(\alpha_1 \ \& \ \alpha_2) \Rightarrow (\alpha_1 \ \& \ \alpha_2) \, U_A \, \beta] \ \supset \qquad \text{(IEISimp2)}$$

$$[\alpha_1 \ \& \ \alpha_2 \Rightarrow \alpha_2 \, U_A \, \beta]$$

are valid.

Proof IEISimp1 is intuitively true because if α_2 is never true, then just α_1, rather than $(\alpha_1 \vee \alpha_2)$ will be waiting for β in the until-after. First the validity of the implication

$$[\Box \neg \alpha_1] \ \& \ [(\alpha_1 \vee \alpha_2) \supset (\alpha_1 \vee \alpha_2) \, U_A \, \beta] \ \supset \ [(\alpha_1 \vee \alpha_2) \supset \alpha_2 \, U_A \, \beta] \quad \text{(A.2)}$$

will be shown. Taking μ of the antecedent, we obtain $\mathbf{A}$.

$$\mathbf{A} \ \equiv \ \forall i \geq 0.\mu[\![\neg \alpha_1]\!](\bar{\sigma}^{(i)}) \ \& \ (\neg \mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}) \vee \forall i \geq 0.\mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(i)})$$

$$\vee \ \exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)}) \ \& \ \forall j, 0 \leq j \leq i.\mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(j)})$$

Let

$$\psi_0 \equiv \forall i \geq 0.\mu[\![\neg \alpha_1]\!](\bar{\sigma}^{(i)})$$

$$\psi_1 \equiv \neg \mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma})$$

$$\psi_2 \equiv \forall i \geq 0.\mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(i)})$$

$$\psi_3 \equiv \exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)}) \ \& \ \forall j, 0 \leq j \leq i.\mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(j)})$$

Now

$$A \equiv \psi_0 \,\&\, (\psi_1 \vee \psi_2 \vee \psi_3)$$

$$\equiv (\psi_0 \,\&\, \psi_1) \vee (\psi_0 \,\&\, \psi_2) \vee (\psi_0 \,\&\, \psi_3). \tag{A.1}$$

The goal is to eliminate α_1 from ψ_2 and ψ_3. Consider first the term $\psi_0 \,\&\, \psi_2$.

$$\psi_0 \,\&\, \psi_2 \equiv \forall i{\ge}0.\neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,\&\, \forall i{\ge}0.\mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(i)})$$

Applying the predicate calculus equivalence

$$(\forall i.P(i)) \,\&\, (\forall i.Q(i)) \equiv \forall i.(P(i) \,\&\, Q(i)) \tag{PredC2}$$

yields

$$\psi_0 \,\&\, \psi_2 \equiv \forall i{\ge}0.[\, \neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,\&\, \mu[\![\alpha_1 \vee \alpha_2]\!](\bar{\sigma}^{(i)}) \,]$$

$$\equiv \forall i{\ge}0.[\, \neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,\&\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,] \tag{A.2}$$

$$\vee\, [\, \neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,\&\, \mu[\![\alpha_2]\!](\bar{\sigma}^{(i)}) \,]$$

The first of the two terms above in formula A.2 is a contradiction, therefore

$$\psi_0 \,\&\, \psi_2 \equiv \forall i{\ge}0.[\, \neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,\&\, \mu[\![\alpha_2]\!](\bar{\sigma}^{(i)}) \,]$$

$$\equiv \forall i{\ge}0.[\, \neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(i)}) \,] \,\&\, \forall i{\ge}0.[\, \mu[\![\alpha_2]\!](\bar{\sigma}^{(i)}) \,]$$

and by PredC2 again

$$\equiv \psi_0 \,\&\, \psi_2'.$$

Now consider the term $\psi_0 \,\&\, \psi_3$ of formula A.1.

$$\psi_0 \;\&\; \psi_3 \equiv \forall i \geq 0.\mu[\![\alpha_1]\!](\bar{\sigma}^{(i)})$$

$$\& \;\exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)}) \;\&\; \forall j, 0 \leq j \leq i.[\; \mu[\![\alpha_1]\!](\bar{\sigma}^{(j)}) \vee \mu[\![\alpha_2]\!](\bar{\sigma}^{(j)}) \;]$$

Now, making use of the predicate calculus theorem which states

$$P \;\&\; (\forall i.Q(i)) \equiv \forall i.(P \;\&\; Q(i)) \qquad \text{(PredC3)}$$

where i does not occur free in P, we change the bound variable i in ψ_0 to k and apply it, giving

$$\psi_0 \;\&\; \psi_3 \equiv \exists i \geq 0.[\; \forall k \geq 0.\neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(k)}) \;] \;\&\; \forall j, 0 \leq j \leq i.\mu[\![\alpha_1]\!](\bar{\sigma}^{(j)}) \vee \mu[\![\alpha_2]\!](\bar{\sigma}^{(j)})$$

$$\equiv \exists i \geq 0.[\; \forall k \geq 0.\neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(k)}) \;] \;\&\; \bigwedge_{j=0}^{i} \mu[\![\alpha_1]\!](\bar{\sigma}^{(j)}) \vee \mu[\![\alpha_2]\!](\bar{\sigma}^{(j)})$$

Using the simplification

$$\neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(j)}) \;\&\; (\mu[\![\alpha_1]\!](\bar{\sigma}^{(j)}) \vee \mu[\![\alpha_2]\!](\bar{\sigma}^{(j)})) \equiv \neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(j)}) \;\&\; \mu[\![\alpha_2]\!](\bar{\sigma}^{(j)})$$

$i+1$ times, we obtain

$$\psi_0 \;\&\; \psi_3 \equiv \exists i \geq 0.[\; \forall k \geq 0.\neg\, \mu[\![\alpha_1]\!](\bar{\sigma}^{(k)}) \;] \;\&\; \forall j, 0 \leq j \leq i.\mu[\![\alpha_2]\!](\bar{\sigma}^{(j)})$$

$$\equiv \psi_0 \;\&\; \psi_3'$$

Since $\psi_0 \;\&\; \psi_2 \equiv \psi_0 \;\&\; \psi_2'$ and $\psi_0 \;\&\; \psi_3 \equiv \psi_0 \;\&\; \psi_3'$ formula A.1 is equivalent to $\psi_0 \;\&\; (\psi_1 \vee \psi_2' \vee \psi_3')$ which is $\mu[\![\alpha_1 \vee \alpha_2 \supset \alpha_2 \; U_A \; \beta]\!](\bar{\sigma})$. Now the validity of formula A.2 is established. Implication IEISimp1 follows by the $\Box\;\Box$ Rule, yielding

$$\vdash \Box([\,\Box \neg\, \alpha_1] \;\&\; [(\alpha_1 \vee \alpha_2) \supset (\alpha_1 \vee \alpha_2) \; U_A \; \beta]) \;\supset\; \Box[(\alpha_1 \vee \alpha_2) \supset \alpha_2 \; U_A \; \beta]$$

and then distribing $\Box$ over $\&$ and reducing $\Box\;\Box\,\alpha_1$ to $\Box\,\alpha_1$.

Implication IEISimp2 is proved similarly.

•

Theorem 4.3. The implication

$$\vdash [\Box\alpha] \ \& \ [\beta \Rightarrow \beta\, U_A\, (\neg\alpha)] \ \supset \ [\beta \Rightarrow \Box\beta] \qquad \text{(SImp)}$$

is valid.

Proof Intuitively, β is waiting for $\neg\alpha$ to occur but it never will occur because $\Box\alpha$. The implication is proved in the semantics. The initial assumptions are $\Box\alpha$ and $\beta \supset \beta\, U_A\, (\neg\alpha)$. Taking μ of the latter yields

$$\mu[\![\beta]\!](\bar{\sigma}) \ \supset \ \mu[\![\beta\, U_A \neg\alpha]\!](\bar{\sigma}). \qquad \text{(A.3)}$$

If $\mu[\![\beta]\!](\bar{\sigma})$ is false then μ of the consequent of implication SImp is trivially true. Assume that $\mu[\![\beta]\!](\bar{\sigma})$ is true then for any state sequence $\bar{\sigma}$. Then by formula A.3 and modus ponens it may be assumed that $\mu[\![\beta\, U_A \neg\alpha]\!]$. Now, by the definition of U_A,

$$\begin{aligned}\mu[\![\beta\, U_A \neg\alpha]\!](\bar{\sigma}) \ &\equiv \ \forall i{\geq}0.\mu[\![\beta]\!](\bar{\sigma}^{(i)}) \\ &\qquad \vee\ \exists i{\geq}0.\mu[\![\neg\alpha]\!](\bar{\sigma}^{(i)}) \ \& \ \forall j, 0{\leq}j{\leq}i.\mu[\![\beta]\!](\bar{\sigma}^{(j)}) \\ &\equiv \ \psi_1 \vee \psi_2\end{aligned}$$

Taking μ of the other assumption, $\Box\alpha$, yields

$$\mu[\![\Box\alpha]\!](\bar{\sigma}) \ \equiv \ \forall i{\geq}0.\mu[\![\alpha]\!](\bar{\sigma}^{(i)})$$

$$\equiv \neg \exists i \geq 0.\mu[\![\neg \alpha]\!](\bar{\sigma}^{(i)}) \tag{A.4}$$

By the theorem PredC1 above ψ_2, which is

$$\exists i \geq 0.[\ \mu[\![\neg \alpha]\!](\bar{\sigma}^{(i)})\ \&\ \forall j, 0 \leq j \leq i.\mu[\![\beta]\!](\bar{\sigma}^{(j)})\ ]$$

implies

$$\exists i \geq 0.[\ \mu[\![\neg \alpha]\!](\bar{\sigma}^{(i)})\ ]\ \ \&\ \ \exists i \geq 0.[\ \forall j, 0 \leq j \leq i.\mu[\![\beta]\!](\bar{\sigma}^{(j)})\ ]. \tag{A.5}$$

But formula A.5 contradicts formula A.4 so ψ_2 cannot be true. This means ψ_1 is then true since it is true that $\psi_1 \vee \psi_2$ and $\neg \psi_2$. ψ_1 is the μ-translation of $\Box \beta$ so it has been shown that $\mu[\![\beta \supset \Box \beta]\!](\bar{\sigma})$.

Thus the validity of the implication

$$\vdash \Box \alpha \ \&\ \beta \supset \beta\, \mathcal{U}_A\, (\neg \alpha)\ \ \supset\ \ \beta \supset \Box \beta \tag{A.6}$$

has been shown. Just as above for the input elimination rules, the entailment version of the rule is justified by using $\Box\ \Box$ Rule on formula A.6, distributing the $\Box$ over the $\&$, and eliminating the redundant $\Box$.

•

Theorem 4.4. The implication

$$\beta_1\ \ \&\ \ \beta_2\ \ \&\ \ [\beta_1 \Rightarrow \beta_1\, \mathcal{U}_A\, (\neg \beta_2)]\ \ \& \tag{A.7}$$

$$[\beta_2 \Rightarrow \beta_2\, \mathcal{U}_A\, (\neg \beta_1)]\ \ \supset\ \ \Box(\beta_1\ \&\ \beta_2)$$

is valid.

Proof Intuitively, β_1 is waiting for $\neg \beta_2$ to become true and β_2 is waiting for $\neg \beta_1$ to become true. Since β_1 and β_2 are initially true, they will wait for each other forever.

We begin the proof in the deductive system then switch o the semantics. With the assumptions $\beta_1 \Rightarrow \beta_1 U_A (\neg \beta_2)$ and $\beta_2 \Rightarrow \beta_2 U_A (\neg \beta_1)$ by $\Box E$ it is known that $\vdash \beta_1 \supset \beta_1 U_A (\neg \beta_2)$ and $\vdash \beta_2 \supset \beta_2 U_A (\neg \beta_1)$. By the assumption $\beta_1 \,\&\, \beta_2$ and modus ponens,

$$G \vdash \beta_1 U_A (\neg \beta_2) \,\&\, \beta_2 U_A (\neg \beta_1)$$

follows where G is the three assumptions. Then by RDED

$$\vdash G \;\supset\; \beta_1 U_A (\neg \beta_2) \,\&\, \beta_2 U_A (\neg \beta_1)$$

follows. The implication

$$\vdash \beta_1 U_A (\neg \beta_2) \,\&\, \beta_2 U_A (\neg \beta_1) \;\supset\; \Box(\beta_1 \,\&\, \beta_2) \tag{A.8}$$

is left to be shown. This will be done in the semantics. Expanding the μ-translations of the two until-after formulas over the $\&$-operator yields

$$\begin{aligned}
&\mu[\![\beta_1 U_A (\neg \beta_2)]\!](\bar{\sigma}) \,\&\, \mu[\![\beta_2 U_A (\neg \beta_1)]\!](\bar{\sigma}) \\
\equiv\; & [\forall i \geq 0.\mu[\![\beta_1]\!](\bar{\sigma}^{(i)}) \;\&\; \forall i \geq 0.\mu[\![\beta_2]\!](\bar{\sigma}^{(i)})] \\
& \vee [\forall i \geq 0.\mu[\![\beta_1]\!](\bar{\sigma}^{(i)}) \;\&\; \exists i \geq 0.\mu[\![\neg \beta_1]\!](\bar{\sigma}^{(i)}) \,\&\, H(\beta_2, i)] \\
& \vee [\exists i \geq 0.\mu[\![\neg \beta_2]\!](\bar{\sigma}^{(i)}) \,\&\, H(\beta_1, i) \;\&\; \forall i \geq 0.\mu[\![\beta_2]\!](\bar{\sigma}^{(i)})] \\
& \vee [\exists i \geq 0.\mu[\![\neg \beta_2]\!](\bar{\sigma}^{(i)}) \,\&\, H(\beta_1, i)] \;\&\; [\exists i \geq 0.\mu[\![\neg \beta_1]\!](\bar{\sigma}^{(i)}) \,\&\, H(\beta_2, i)] \\
\equiv\; & \psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4
\end{aligned}$$

where

$$H(\beta,i) \equiv \forall j, 0 \le j \le i. \mu[\![\beta]\!](\bar{\sigma}^{(j)})$$

The subformulas ψ_2, ψ_3, and ψ_4 will be proved false, and this leaves only ψ_1, which is the translation of the consequent of (A.8).

To see that ψ_2 is false, let $\psi_{21} \equiv \forall i \ge 0. \mu[\![\beta_1]\!](\bar{\sigma}^{(i)})$. Then

$$\psi_2 \equiv \psi_{21} \ \& \ \exists i \ge 0. [\ \mu[\![\neg \beta_1]\!](\bar{\sigma}^{(i)}) \ \& \ H(\beta_2,i)\ ].$$

By the theorem of predicate calculus PredC1,

$$\exists i \ge 0. [\ \mu[\![\neg \beta_1]\!](\bar{\sigma}^{(i)}) \ \& \ H(\beta_2,i)\ ]$$

$$\equiv \ \exists i \ge 0. [\ \mu[\![\neg \beta_1]\!](\bar{\sigma}^{(i)})\ ] \ \& \ \exists i \ge 0. [\ H(\beta_2,i)\ ],$$

and by the PC tautology

$$(u \ \& \ v) \ \& \ (v \supset w) \supset (u \ \& \ w) \qquad \text{(PC3)}$$

it follows that

$$\psi_2 \supset \psi_{21} \ \& \ \exists i \ge 0. [\ \mu[\![\neg \beta_1]\!](\bar{\sigma}^{(i)})\ ] \ \& \ \exists i \ge 0. [\ H(\beta_2,i)\ ]$$

$$\equiv \psi_{21} \ \& \ (\neg \forall i \ge 0. \mu[\![\beta_1]\!](\bar{\sigma}^{(i)})) \ \& \ (\exists i \ge 0. [\ H(\beta_2,i)])$$

$$\equiv \psi_{21} \ \& \ \psi_{22} \ \& \ \psi_{23}.$$

Notice now that ψ_{21} and ψ_{22} contradict each other. It follows then that ψ_2 is false.

ψ_3 is proven false similarly.

ψ_4 is proven false by a double induction on the two time indices associated with each of its two constituent clauses. Let ψ_4 be written

$$\psi_4 \equiv \exists i \geq 0.\phi_1(i) \ \& \ \exists i \geq 0.\phi_2(i)$$

where

$$\phi_1(i) \equiv \mu[\![\neg \beta_2]\!](\bar{\sigma}^{(i)}) \ \& \ H(\beta_1, i)$$

and

$$\phi_2(i) \equiv \mu[\![\neg \beta_1]\!](\bar{\sigma}^{(i)}) \ \& \ H(\beta_2, i).$$

Proving that ψ_4 is false is equivalent to proving ξ where

$$\xi \equiv \neg \left[\bigvee_{i=0}^{\infty} \phi_1(i) \ \& \ \bigvee_{j=0}^{\infty} \phi_2(j) \right].$$

Let

$$\xi(k,l) \equiv \neg \left[\bigvee_{i=0}^{k} \phi_1(i) \ \& \ \bigvee_{j=0}^{l} \phi_2(j) \right] \textit{ for all } k, l.$$

Proving $\forall k \geq 0. \forall l \geq 0. \xi(k,l)$ will suffice to show ξ. The proof of $\forall k \geq 0. \forall l \geq 0. \xi(k,l)$ is a double induction proof. This consists of proving by single induction on k that $\forall l \geq 0. \xi(k,l)$, which is again proved by single induction on l.

k-basis Prove $\forall l \geq 0. \xi(0,l)$ by induction on l.

l-basis $\xi(0,l)$ evaluated at $l=0$ is

$$\xi(0,0) \equiv \neg [\ \phi_1(0) \ \& \ \phi_2(0)\ ]$$

$$\equiv \neg \left[\mu[\![\neg \beta_2]\!](\bar{\sigma}^{(0)}) \ \& \ \mu[\![\beta_1]\!](\bar{\sigma}^{(0)}) \ \& \ \mu[\![\neg \beta_1]\!](\bar{\sigma}^{(0)}) \ \& \ \mu[\![\beta_2]\!](\bar{\sigma}^{(0)}) \right]$$

$\equiv$ **true.**

Clearly $\xi(0,0)$ is true.

l-induction Assume an induction hypothesis of $\xi(0,l)$. Expanding $\xi(0,l+1)$,

$$\xi(0,l+1) \equiv \neg\left[\phi_1(0)\ \&\ \bigvee_{i=0}^{l+1}\phi_2(i)\right]$$

$$\equiv \neg\left[\phi_1(0)\ \&\ \bigvee_{l=0}^{l}\right]\ \&\ \neg\left[\phi_1(0)\ \&\ \phi_2(l+1)\right]$$

$$\equiv \xi(0,l)\ \&\ \neg\left[\phi_1(0)\ \&\ \phi_2(l+1)\right].$$

$\xi(0,l)$ is the induction hypothesis and is therefore true, so $\neg[\phi_1(0)\ \&\ \phi_2(l+1)]$ remains to be shown. Since $\phi_1(0) \supset \mu[\![\beta_2]\!](\bar{\sigma}^{(0)})$ and $\phi_2(l+1) \supset \neg\,\mu[\![\beta_2]\!](\bar{\sigma}^{(0)})$, a contradiction exists. Therefore $\neg[\phi_1(0)\ \&\ \phi_2(l+1)]$ is true. At this point $\forall l{\ge}0.\xi(0,l)$ has been proved, the basis of the induction on k.

k-induction Assume an induction hypothesis of $\forall l{\ge}0.\xi(k,l)$. $\forall l{\ge}0.\xi(k+1,l)$ is to be proved. Expanding $\forall l{\ge}0.\xi(k+1,l)$,

$$\forall l{\ge}0.\xi(k+1,l) \equiv \forall l{\ge}0.\neg\left[\bigvee_{i=0}^{k+1}\phi_1(i)\ \&\ \bigvee_{j=0}^{l}\phi_2(j)\right]$$

$$\equiv \forall l{\ge}0.\neg\left[(\bigvee_{i=0}^{k}\phi_1(i)\ \&\ \bigvee_{j=0}^{l}\phi_2(j)) \vee (\phi_1(k+1)\ \&\ \bigvee_{j=0}^{l}\phi_2(j))\right]$$

$$\equiv \forall l{\ge}0.\neg\left[\bigvee_{i=0}^{k}\phi_1(i)\ \&\ \bigvee_{j=0}^{l}\phi_2(j)\right] \quad (A.9)$$

$$\& \ \forall l \geq 0. \neg \left[\phi_1(k+1) \ \& \ \bigvee_{j=0}^{l} \phi_2(j) \right]$$

The first factor of (A.9) is the induction hypothesis and is therefore true. The second is left to be proved. It will be proved by induction on l.

l-basis $\neg [\phi_1(k+1) \ \& \ \bigvee_{j=0}^{l} \phi_2(j)]$ evaluated at $l=0$ is $\neg [\phi_1(k+1) \ \& \ \phi_2(0)]$. Since $\phi_1(k+1) \supset \mu[\![\beta_1]\!](\bar{\sigma}^{(0)})$ and $\phi_2(0) \supset \neg \mu[\![\beta_1]\!](\bar{\sigma}^{(0)})$, a contradiction exists. Therefore $\neg [\phi_1(k+1) \ \& \ \phi_2(0)]$ is true.

l-induction Assume an induction hypothesis of $\neg [\phi_1(k+1) \ \& \ \bigvee_{j=0}^{l} \phi_2(j)]$. Then

$$\neg \left[\phi_1(k+1) \ \& \ \bigvee_{j=0}^{l+1} \phi_2(j) \right]$$

$$\equiv \neg \left[\phi_1(k+1) \ \& \ \bigvee_{j=0}^{l} \phi_2(j) \right] \ \& \ \neg \left[\phi_1(k+1) \ \& \ \phi_2(l+1) \right]. \qquad \text{(A.10)}$$

The first factor in (A.10) is the induction hypothesis. The second factor remains to be proved.

case $k>l$. Then $k+1>l+1$ and there is a $j = l+1$ in the range $0 \leq j \leq k+1$ such that $\phi_1(k+1) \supset \mu[\![\beta_1]\!](\bar{\sigma}^{(j)})$ and $\phi_2(l+1) \supset \neg \mu[\![\beta_1]\!](\bar{\sigma}^{j})$, a contradiction. Therefore $\neg [\phi_1(k+1) \ \& \ \phi_2(l+1)]$.

case $k \leq l$. Then $k+1 \leq l+1$ and there is a $j = k+1$ in the range $0 \leq j \leq l+1$ such that $\phi_1(k+1) \supset \mu[\![\beta_2]\!](\bar{\sigma}^{(j)})$ and $\phi_2(l+1) \supset \neg \mu[\![\beta_2]\!](\bar{\sigma}^{j})$, a contradiction. Therefore $\neg [\phi_1(k+1) \ \& \ \phi_2(l+1)]$.

This concludes the induction on l and induction on k so $\forall k \geq 0. \forall l \geq 0. \xi(k,l)$ has been proved. This concludes the proof that ψ_4 is false. Since ψ_2, ψ_3, and ψ_4 are false and

$$\psi_1 \vee \psi_2 \vee \psi_3 \vee \psi_4$$

is true, ψ_1 must be true. Since

$$\psi_1 \equiv \mu[\![\Box \beta_1 \,\&\, \Box \beta_2]\!](\bar{\sigma})$$

$$\equiv \mu[\![\Box (\beta_1 \,\&\, \beta_2)]\!](\bar{\sigma})$$

the validity of implication (A.8) is established.

Implication (A.7) can be obtained by noting that

$$\vdash \beta_1 \,\&\, \beta_2 \,\&\, [\beta_1 \Rightarrow \beta_1 \, U_A \, (\neg \beta_2)] \,\&\, [\beta_2 \Rightarrow \beta_2 \, U_A \, (\neg \beta_1)] \supset$$

$$\beta_1 \, U_A \, (\neg \beta_2) \,\&\, \beta_2 \, U_A \, (\neg \beta_1)$$

which is the antecedent of the implication A.8.

•

Theorem 4.5. The implication

$$\vdash \beta_1 \,\&\, \beta_2 \,\&\, [\beta_1 \Rightarrow \beta_1 \, U_A \, (\neg \beta_2)] \,\& \qquad \text{(A.11)}$$

$$\beta_2 \Rightarrow \beta_2 \, U_A \, (\neg \beta_1 \,\&\, \alpha)] \supset \Box (\beta_1 \,\&\, \beta_2)$$

is valid.

Proof Intuitively, β_1 is waiting for $\neg \beta_2$ to become true and β_2 is waiting for $(\neg \beta_1 \,\&\, \alpha)$ to become true. Since β_1 and β_2 are initially true, β_1 will be waiting forever for $\neg \beta_2$ to happen. β_2 will be waiting for $\neg \beta_1$ to happen to make $(\neg \beta_1 \,\&\, \alpha)$ true. The implication (A.11) can be proved just like implication A.9 was proved, the

only modification being that " $\&\ \alpha$" is carried along throughout.

•

Theorem 4.6. The implication

$$\nabla(\beta_1 \,\&\, \beta_2) \;\&\; [\beta_1 \Rightarrow \beta_1\, U_A\, (\neg\, \beta_2)] \;\& \qquad \text{(EIGSimp)}$$

$$[\beta_2 \Rightarrow \beta_2\, U_A\, (\neg\, \beta_1 \,\&\, \alpha)] \;\supset\; \nabla\, \Box(\beta_1 \,\&\, \beta_2)$$

is valid.

Proof Starting with the antecedent of the of EIGSimp, using theorem T33 of Section 4.1.8 twice yields

$$\vdash \nabla(\beta_1 \,\&\, \beta_2) \,\&\, [\beta_1 \Rightarrow \beta_1\, U_A\, (\neg\, \beta_2)] \,\&\, [\beta_2 \Rightarrow \beta_2\, U_A\, (\neg\, \beta_1 \,\&\, \alpha)] \;\supset \quad \text{(A.12)}$$

$$\nabla((\beta_1 \,\&\, \beta_2) \,\&\, [\beta_1 \Rightarrow \beta_1\, U_A \neg\, \beta_2)] \,\&\, [\beta_2 \Rightarrow \beta_2\, U_A\, (\neg\, \beta_1 \,\&\, \alpha)])$$

Using $\nabla\ \nabla$ Rule, formula A.11 becomes an implication whose antecedent is in the desired form.

$$\vdash \nabla((\beta_1 \,\&\, \beta_2) \,\&\, [\beta_1 \Rightarrow \beta_1\, U_A\, (\neg\, \beta_2)] \,\&\, [\beta_2 \Rightarrow \beta_2\, U_A\, (\neg\, \beta_1 \,\&\, \alpha)]) \;\supset \quad \text{(A.13)}$$

$$\nabla\, \Box(\beta_1 \,\&\, \beta_2)$$

Transitivity of implication on formulas (A.12) and (A.13) gives us EIGSimp.

•

Theorem 4.7. The implication

$$\vdash ((\alpha\, U_A\, \beta)\ \&\ \nabla\beta) \supset \nabla(\alpha\ \&\ \beta) \qquad \text{(CRimp)}$$

is valid.

Proof The definition of U_A says that either (i) α is true forever or (ii) β will eventually be true and at some point α and β are true together. These two cases correspond to ψ_1 and ψ_2 below being true.

$$\mu[\![w_1\, U_A\, w_2]\!](\bar{\sigma}) \equiv \forall i \geq 0.\mu[\![\alpha]\!](\bar{\sigma})$$

$$\vee\ \exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)})\ \&\ \forall j, 0 \leq j \leq i.\mu[\![\alpha]\!](\bar{\sigma}^{(j)})$$

$$\equiv \psi_1 \vee \psi_2$$

case ψ_1. $\mu[\![\nabla\beta]\!](\bar{\sigma}) \equiv \exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)})$. Using the predicate calculus theorem

$$(\forall i.P(i))\ \&\ (\exists i.Q(i)) \supset \exists i.(P(i)\ \&\ Q(i))$$

on $\psi_1\ \&\ \mu[\![\nabla\beta]\!](\bar{\sigma})$ we obtain $\exists i \geq 0.\mu[\![\alpha]\!](\bar{\sigma}^{(i)})\ \&\ \mu[\![\beta]\!](\bar{\sigma}^{(i)})$ which is $\mu[\![\nabla(\alpha\ \&\ \beta)]\!](\bar{\sigma})$.

case ψ_2.

$$\psi_2 \equiv \exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)})\ \&\ \forall j, 0 \leq j \leq i.\mu[\![\alpha]\!](\bar{\sigma}^{(j)})$$

$$\equiv \exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)})\ \&\ [\ \mu[\![\alpha]\!](\bar{\sigma}^{(0)})\ \&\ \cdots\ \&\ \mu[\![\alpha]\!](\bar{\sigma}^{(i-1)})\ \&\ \mu[\![\alpha]\!](\bar{\sigma}^{(i)})\ ]$$

Applying the theorem PredC1 we obtain

$$\exists i \geq 0.\mu[\![\beta]\!](\bar{\sigma}^{(i)})\ \&\ \exists i \geq 0.[\ \mu[\![\alpha]\!](\bar{\sigma}^{(0)})\ \&\ \cdots\ \&\ \mu[\![\alpha]\!](\bar{\sigma}^{(i-1)})\ ]$$

which implies

$$\exists i \geq 0 . \mu[\![\beta]\!](\bar{\sigma}^{(i)}) \ \&\ \mu[\![\alpha]\!](\bar{\sigma})$$

which is $\mu[\![\nabla (\alpha \ \&\ \beta)]\!](\bar{\sigma})$. •

REFERENCES

[Arm69] D. B. Armstrong, A. D. Friedman, P. R. Menon. "Design of asynchronous circuits assuming unbounded gate delays." *IEEE Transactions on Computers C-18,* 12 (December 1969), 1110-1120.

[Bac78] J. Backus. "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs." *Communications of the ACM,* Vol. 21, No. 8, 613-641, August 1978.

[Bar81] M. R. Barbacci. "Instruction Set Processor Specifications (ISPS): The notation and its applications." *IEEE Transactions on Computers C-30,* 1 (January 1981), 24-40.

[Bar83] H. G. Barrow. Proving the correctness of digital hardware designs. Proceedings of the National Conference on Artificial Intelligence, Washington D.C., August 22-26, 1983.

[Ber82] H. K. Berg, W. E. Boebert, W. R. Franta, T. G. Moher. *Formal Methods of Program Verification and Specification.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[Boc82] G. V. Bochmann. "Hardware specification with temporal logic: An example." *IEEE Transactions on Computers C-31,* 3 (March 1982), 223-231.

[Bos84] P. G. Bosco, G. Giandonato, E. Giovannetti, "A PROLOG system for the verification of concurrent processes against temporal logic specifications." Proceedings of the Second International Logic Programming Conference, Uppsala, Sweden, 1984, pages 219-229.

[Cla83] E. M. Clarke, E. A. Emerson, A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. Department of Computer Science, Carnegie-Mellon University, Report No. CMU-CS-83-152, 1983.

[Dar81] J. A. Darringer, W. H. Joyner Jr., C. L. Berman, and L. Trevillyan. "Logic synthesis through local transformations." *IBM Journal of Research and Development 25,* 4 (July 1981), 272-280.

[deB80] J. de Bakker. *Mathematical Theory of Program Correctness.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1980.

[Dit82] D. R. Ditzel, program chairman. Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems, Palo Alto, California, May, 1982. Published as *SIGARCH Computer Architecture News 10,* 2 (March 1982).

[Dud83] S. Dudani and E. Stabler. Types of Hardware Description. Proceedings of

the IFIP WG 10.2 Sixth International Symposium on Computer Hardware Description Languages and their Applications, Pittsburgh, Pennsylvania, May, 1983, pages 127-136.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1982.

[Flo67] R. W. Floyd. "Assigning meanings to programs." Proceedings of the American Math. Society Symposia on Applied Mathematics, Vol. 19, 19-31, 1967.

[Fri75] A. D. Friedman and P. R. Menon. *Theory and Design of Switching Circuits.* Computer Science Press, Inc., Rockville, Maryland, 1975.

[Gab80] D. Gabbay, A. Pnueli, S. Shelah, J. Stavi. On the temporal analysis of fairness. Proceedings of the 7th ACM Symposium on the Principles of Programming Languages, Las Vegas, Nevada, January, 1980, pages 163-173.

[Goo70] D. I. Good. Toward a man-machine system for proving program correctness. Computation Center, University of Texas, Austin, Report No. TSN-11, 1970.

[Gor81a] M. Gordon. A model of register transfer systems with applications to microcode and VLSI correctness. Department of Computer Science, University of Edinburgh, 1981.

[Gor81b] M. Gordon. Register transfer systems and their behavior. Proceedings of the IFIP WG 10.2 Fifth International Conference on Computer Hardware Description Languages and their Applications, Kaiserslautern, West Germany, September, 1981, pages 23-36.

[Gor81c] M. Gordon. A very simple model of sequential behavior of nMOS. In J. P. Gray, editor, *VLSI 81*, pages 85-94, Academic Press, 1981.

[Har79] D. Harel. *First-Order Dynamic Logic.* Springer-Verlag, Berlin, 1979. *Lecture Notes in Computer Science* Number 68.

[Hai82] B. T. Hailpern. *Verifying Concurrent Processes Using Temporal Logic.* Springer-Verlag, Berlin, 1982.

[Hol82] L. A. Hollaar. "Direct implementation of asynchronous control units." *IEEE Transactions on Computers C-31*, 12 (December 1982), 1133-1141.

[Hoa69] C. A. R. Hoare. "An axiomatic basis for computer programming." *Communications of the ACM*, Vol. 12, No. 10, 576-583, October 1969.

[Hoa78] C. A. R. Hoare. "Communicating Sequential Processes." *Communications of the ACM 21*, 8 (August 1978), 666-677.

[Hug68] G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic.* Methuen and Co., Ltd., London, 1968.

[Iga75] S. Igarashi, R. L. London, and D. C. Luckham. "Automatic program verification I: A logical basis and its implementation." *Acta Informatica 4,* 145-182, 1975.

[Kar83] A. R. Karlin, H. W. Trickey, J. D. Ullman. Experience with a regular expression compiler. Department of Computer Science, Stanford University, Report No. STAN-CS-83-972, 1983.

[Kle67] S. C. Kleene. *Mathematical Logic.* John Wiley and Sons, Inc., New York, 1967.

[Kro84] F. Kroger. "A generalized nexttime operator in temporal logic." *Journal of Computer and System Sciences 29,* 80-98, 1984.

[Lah81] D. Lahti, *Applications of a Functional Programming Language.* M.S. Thesis. Computer Science Department, University of California, Los Angeles, 1981.

[Lei81] C. E. Leiserson. *Area-Efficient VLSI Computation.* Ph.D. Thesis. Department of Computer Science, Carnegie-Mellon University, 1981.

[Mal81] Y. Malachi and S. S. Owicki. Temporal specifications of self-timed systems. In H. T. Kung, B. Sproul, and G. Steele, editors, *VLSI Systems and Computations,* pages 203-212, Computer Science Press, Inc., Rockville, Maryland, 1981.

[Man72] Z. Manna and J. Vuillemin. "A fixpoint approach to the theory of computation." *Communications of the ACM 15,* 7 (July 1972), 528-536.

[Man81a] Z. Manna and A. Pnueli. Verfication of concurrent programs, part I: the temporal framework. Department of Computer Science, Stanford University, Report No. STAN-CS-81-836 1981.

[Man81b] Z. Manna. Verfication of sequential programs: temporal axiomization. Department of Computer Science, Stanford University, Report No. STAN-CS-81-877, 1981.

[Man82] Z. Manna and A. Pnueli. Verification of concurrent programs: Proving eventualities by well-founded ranking. Department of Computer Science, Stanford University, Report No. STAN-CS-82-915, 1982.

[McF81] M. C. McFarland. *Mathematical Models for Formal Verification in a Design Automation System.* Ph.D. Thesis. Department of Electrical Engineering, Carnegie-Mellon University, 1981.

[Mea80] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1980.

[Mes84] F. Meshkinpour. *On Specification and Design of Digital Systems Using an Applicative Hardware Description Language*. M.S. Thesis. Computer Science Department, University of California, Los Angeles, 1984.

[Mil65] R. E. Miller. *Switching Theory, Volume 2*. Wiley, New York, 1965.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, 1979. *Lecture Notes in Computer Science* Number 92.

[Mil82] G. J. Milne. CIRCAL: A calculus for circuit description. Department of Computer Science, University of Edinburgh, 1982.

[Mil83a] G. J. Milne. The correctness of a simple silicon compiler. Department of Computer Science, University of Edinburgh, 1983.

[Mil83b] G J. Milne. A formal basis for the analysis of circuit timing. Department of Computer Science, University of Edinburgh, 1983.

[Mis83] B. Mishra and E. M. Clarke. Automatic and hierarchical verification of asynchronous circuits using temporal logic. Department of Computer Science, Carnegie-Mellon University, Report No. CMU-CS-83-155, 1983.

[Mos83] B. C. Moszkowski. *Reasoning about Digital Circuits*. Ph.D. Thesis, Department of Computer Science, Standford University, 1983.

[Mul59] D. E. Muller and W. S. Bartky. A theory of switching circuits. Proceedings of an International Symposium on the Theory of Switching, volume 29, Annals of the Computation Laboratory of Harvard University, Harvard Univ. Press, 1959, pages 204-243.

[Owi76] S. S. Owicki and D. Gries. "An axiomatic proof technique for parallel programs I." *Acta Informatica 6*, 319-340, 1976.

[Owi82] S. S. Owicki and L. Lamport. "Proving liveness properties of concurrent programs." *ACM Transactions on Programming Languages and Systems 4*, 3 (July 1982), 455-495.

[Pat76a] D. A. Patterson. "Strum: Structured microprogram development system for correct firmware." *IEEE Transactions on Computers C-25*, 10 (October 1976), 974-985.

[Pat76b] D. A. Patterson. *Verification of Microprograms*. Ph.D. Thesis. Computer Science Department, University of California, Los Angeles, 1976.

[Pit83] V. Pitchumani and E. Stabler. "An inductive assertion method for register transfer level design verification." *IEEE Transactions on Computers C-32*, 12 (December 1983), 1073-1080.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. Proceedings of the Eighteenth Symposium on the Foundations of Computer Science, Providence,

Rhode Island, 1977, pages 46-57.

[Pnu81] A. Pnueli. "The temporal semantics of concurrent programs." *Theoretical Computer Science 13*, 1981, 45-60.

[Res71] N. Rescher and A. Urquart. *Temporal Logic.* Springer-Verlag, New York, 1971.

[Sco71] D. Scott. "The lattice of flow diagrams." In *Symposium on Semantics of Algorithmic Languages*, pages 311-366, Spinger-Verlag, Berlin, 1971. *Lecture Notes in Mathematics* Number 188.

[Sei80a] C. L. Seitz. "System timing," chapter 7 of [Mea80], 218-262.

[Sei80b] C. L. Seitz. "Ideas about arbiters." Lambda, First Quarter, 1980.

[Sto77] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, Massachusetts, 1977.

[Ueh83] T. Uehara, T. Saito, F. Maruyama, and N. Kawato. DDL verifier and temporal logic. Proceedings of the IFIP WG 10.2 Sixth International Symposium on Computer Hardware Description Languages and their Applications, Pittsburgh, Pennsylvania, May, 1983, pages 91-102.

[Ull84] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, Inc., Rockville, Maryland, 1984.

[Ung69] S. H. Unger. *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, New York, 1969.

[Wag77] T. J. Wagner, *Hardware Verification.* Ph.D. Thesis, Department of Computer Science, Stanford University, 1977.

[Wol83] P. Wolper. "Temporal logic can be more expressive." *Information and Control 56*, 72-99, 1983.

[Wol84] Z. Manna and P. Wolper. "Synthesis of communicating processes from temporal logic specifications." *ACM Transactions of Programming Languages and Systems 6*, 1 (January 1984), 68-93.