

Delay Insensitive Protocol
and IC Design

by

Greg F. Soprovich

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the Degree of
Master of Science
in
Electrical Engineering

Winnipeg, Manitoba, 1989
Greg F. Soprovich



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-54840-1

DELAY INSENSITIVE PROTOCOL AND IC DESIGN

BY

GREG F. SOPROVICH

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

© 1989

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this thesis.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

Abstract

The purpose of this thesis is to study a class of logic that has the property of being system clock-free. Independence from system level clocking is provided by relaxing timing requirements so that signals are fundamentally Delay Insensitive. A class of signal specifications, and protocol methods, are provided as a basis for exploring Delay Insensitive logic.

During this project a Delay Insensitive protocol was proposed as a means for providing high level communication between logical blocks. An integrated circuit (IC) was implemented in order to test various Delay Insensitive principles, and to evaluate the overhead associated with adding delay insensitivity to a serial protocol. Overall testing indicated that IC overhead was not excessive given the tasks the asynchronous Delay Insensitive Communication IC implements. Further, the validity of delay insensitivity was demonstrated by the correct logical functioning of the IC despite introduction of significant external delays, and the overall limitations imposed by insufficient off-chip driver circuits.

A Delay Insensitive design methodology is presented and is utilized to describe fully Delay Insensitive representations for the asynchronous protocol. A mixed mode circuit methodology is developed to provide the capability of combining Delay Insensitive logic with locally synchronous devices. The advantage of mixed mode circuits is the capability to extend the lifespan of many current cell libraries within a Delay Insensitive framework. The ability to design within a Delay Insensitive environment appears promising since it could enhance automation of many of the synchronization tasks between major functional blocks of logic.

Acknowledgements

I would like to thank my advisor Professor Robert D. McLeod without whose assistance and encouragement this thesis would not have been possible. Additionally, I would like to thank Dr. Fred U. Rosenberger for providing me with timely reprints and internal memoranda regarding this research area.

I would like to also thank Roland Schneider, Chris Schneider, and Peter Hortensius for their assistance in the VLSI design laboratory. Additional thanks are given to the many coworkers who helped me weather the travails of the Northern Telecom Dracula Design Rule Checker in the VLSI design lab.

Financial support from the University of Manitoba, and equipment loans from the Canadian Microelectronics Corporation are greatly appreciated.

I would also like to thank my family and friends for their support during this project.

TABLE OF CONTENTS

| | |
|---|-----|
| Abstract | 11 |
| Acknowledgements | iii |
| LIST OF Figures | vi |
| LIST OF TABLES | vii |
| CHAPTER 1: DELAY INSENSITIVITY AND DEVICE DESIGN | 1 |
| 1.1 Synchronization and System Behavior | 1 |
| 1.2 Self-Timed Systems | 6 |
| 1.2.1 Definition of Self-Timed Systems | 6 |
| 1.2.2 Requirements for a Self-Timed System | 6 |
| 1.3 Attributes of Self-Timed Systems | 10 |
| CHAPTER 2: DELAY INSENSITIVE SYSTEMS | 15 |
| 2.1 Delay Insensitive Modules | 15 |
| 2.1 System Considerations in DI Modules | 16 |
| 2.1.1 Signal Conventions in DI Modules | 16 |
| 2.1.2 Synchronization Signals | 20 |
| 2.1.3 The Muller C Element (Join) | 22 |
| 2.1.4 Module Forms | 23 |
| 2.2 Delay Insensitive Protocols | 28 |
| 2.3 Design Considerations in DI Systems | 33 |
| CHAPTER 3: AN ASYNCHRONOUS DI INTERFACE IC | 39 |
| 3.1 Implementation Goals | 39 |
| 3.2 Protocol Specifications | 40 |
| 3.3 Physical Implementation | 44 |
| 3.3.1 Input Serial Channel Design | 44 |
| 3.3.2 Output Channel Design | 50 |
| 3.3.3 Muller C Design | 55 |
| 3.3.4 Microprocessor Interface | 57 |
| 3.3.5 Design Area and Layout | 59 |
| 3.4 Design for Testability | 63 |
| 3.5 Functional Simulation Results | 67 |
| 3.5.1 Simulation Procedure | 67 |
| 3.5.2 Simulation Performance | 69 |
| 3.6 Test Results | 71 |
| 3.6.1 Functional Testing | 71 |
| 3.6.2 Dynamic Testing | 73 |
| 3.7 Evaluation | 76 |
| CHAPTER 4: MACROCELL DESIGN | 78 |
| 4.1 Why Another Design Method? | 78 |
| 4.2 A Logic Design Methodology | 79 |
| 4.3 A Hardware Implementation of DI State Equations | 85 |
| 4.3.1 Construction of an EISG Protocol Receiver | 92 |
| 4.3.2 Mixed Mode Design | 101 |

| | |
|---------------------------|-----|
| 4.3.4 System Level Design | 114 |
| CHAPTER 5: SUMMARY | 117 |
| REFERENCES | 119 |

LIST OF FIGURES

| | | |
|------------|---------------------|-----|
| Figure 1: | Alpha Scaling | 4 |
| Figure 2: | Handshaking | 9 |
| Figure 3: | Self Timed Systems | 11 |
| Figure 4: | Isochronic Zones | 12 |
| Figure 5: | DI Modules | 17 |
| Figure 6: | DI Partial Ordering | 19 |
| Figure 7: | 4 Cycle Protocol | 21 |
| Figure 8: | Multiple Joins | 24 |
| Figure 9: | Pipelined STes | 25 |
| Figure 10: | Pipeline Modules | 27 |
| Figure 11: | Parallel STes | 29 |
| Figure 12: | Tri-State Line | 31 |
| Figure 13: | Protocol (1 Rail) | 32 |
| Figure 14: | Protocol (2 Rail) | 34 |
| Figure 15: | Double Rail Data | 35 |
| Figure 16: | DI Modules | 38 |
| Figure 17: | Chip Protocol | 42 |
| Figure 18: | Input Channel | 45 |
| Figure 19: | Input SM | 49 |
| Figure 20: | Output Channel | 52 |
| Figure 21: | Output SM | 54 |
| Figure 22: | Muller C Circuit | 56 |
| Figure 23: | Register Ordering | 58 |
| Figure 24: | SM Observability | 66 |
| Figure 25: | Test System | 75 |
| Figure 26: | ISG of C Element | 81 |
| Figure 27: | EISG of C Element | 83 |
| Figure 28: | C Element Logic | 84 |
| Figure 29: | Q Module | 87 |
| Figure 30: | Q Flop | 89 |
| Figure 31: | Q Module Waveform | 91 |
| Figure 32: | Q Flop Reset | 93 |
| Figure 33: | Two Rail Protocol | 94 |
| Figure 34: | Protocol ISG | 96 |
| Figure 35: | EISG of Receiver | 97 |
| Figure 36: | EISG Logic Map | 99 |
| Figure 37: | DI Receiver | 102 |
| Figure 38: | Pulse Circuit | 105 |
| Figure 39: | Rail Data to Binary | 107 |
| Figure 40: | Binary to Rail Data | 108 |
| Figure 41: | Central DI Stage | 109 |
| Figure 42: | Driver DI Stage | 112 |
| Figure 43: | Driver ISG | 113 |

LIST OF TABLES

| | | |
|-------------|-----------------------------------|----|
| TABLE I: | MULLER C ELEMENT TRUTH TABLE | 22 |
| TABLE II: | INPUT CHANNEL SM SIGNAL FUNCTION | 50 |
| TABLE III: | OUTPUT CHANNEL SM SIGNAL FUNCTION | 53 |
| TABLE IV: | INTERNAL REGISTER CONFIGURATION | 57 |
| TABLE V: | PIN USAGE | 60 |
| TABLE VI: | DIE AREA USAGE | 62 |
| TABLE VII: | STATE TEST ADDRESSES | 65 |
| TABLE VIII: | CHANNEL PERFORMANCE UNDER APLSIM | 70 |

CHAPTER 1: DELAY INSENSITIVITY AND DEVICE DESIGN

1.1 Synchronization and System Behavior

Modern digital systems, both at the integrated circuit level and higher levels, require very stringent synchronization to satisfy basic operating characteristics. In particular, the distribution constraints of clock signals and synchronization of parallel data along long data paths can be particularly difficult to satisfy in system specifications. In this thesis, system will refer to the transistor/logic networks common to all digital and computer organizations, whether on a single chip or spread across many boards. The scope of the thesis is based upon the effect that delay has upon synchronization and control signals. It is the act of synchronizing signals, and signal ordering over long paths, that provides one of modern system designs greatest challenges now, and in the future.

In any significant modern system, the actual wires that interconnect switching elements and operational blocks are far from the idealized versions represented in elementary switching theory and logic. Instead, on-chip wires generally have significant amounts of resistance and capacitance distributed

along their length, and wires at the board level have resistance, capacitance, and inductance along their length. The result is that transistor switching speed is significantly retarded by long wires both on and off-chip. Further, significant delays in a signal can often be introduced by the line itself since the signal must physically propagate down the highly non-ideal transmission line. Modern integrated circuits (ICs) have reached the point where the switching speed of individual transistors is no longer significantly greater than the transmission line delay of a long wire. Thus, logic designers may no longer discount the wire length and characteristics when designing logic circuits in silicon, or at a board level. Of course, this problem is not new; system designers have long contended with the delay problem when designing major computers since even TTL can switch much faster than most card cage delays in such systems. A good example of this characteristic is the VAX 750 family which exists on a 1000 nS bus at the card cage level, while having much greater card-level performance. Thus, line delay is an increasingly pervasive problem at all levels of system design.

How has technology influenced the problem of transmission line delay? Primarily, the advance of technology has only made the problem much more difficult to approach. It is true that great effort has been made to reduce wire capacitance, inductance and resistance by using high density packing, improved board materials, and so on, but these remedies only

moderately alleviate the problem. While progress was made in board layout and materials, integration has correspondingly resulted to smaller active devices with substantially shorter switching times. As switching speeds continue to increase, the significance of line delay induced signal skew in clock, data and control signals becomes much more crucial. For example, Figure 1 shows an example of scaling a circuit down by a factor of (A) while an input wire delay mismatch stays essentially unscaled along its length. If wire delay were to stay constant, the ratio between the switching time and the delay is obviously much smaller -- in fact, the delay may have become more significant, or even critical.

Unfortunately, transmission line delay is a much greater problem than it would superficially appear to be. After all, if the line delay increases, why not simply insert delays, as needed, in order to assure proper synchronization? This is often exactly what is done. The difficulty with such an approach is that variations in the fabrication process make it very difficult to accurately estimate line delay. If we must insert a large number of delay lines we will likely see such adjustments reflected in reductions in device yields. Furthermore, clock distribution over a chip becomes much more complex if proper IC operation depends on the insertion of many carefully tailored delays between functional blocks. Such design methodologies can be very arduous and time consuming,

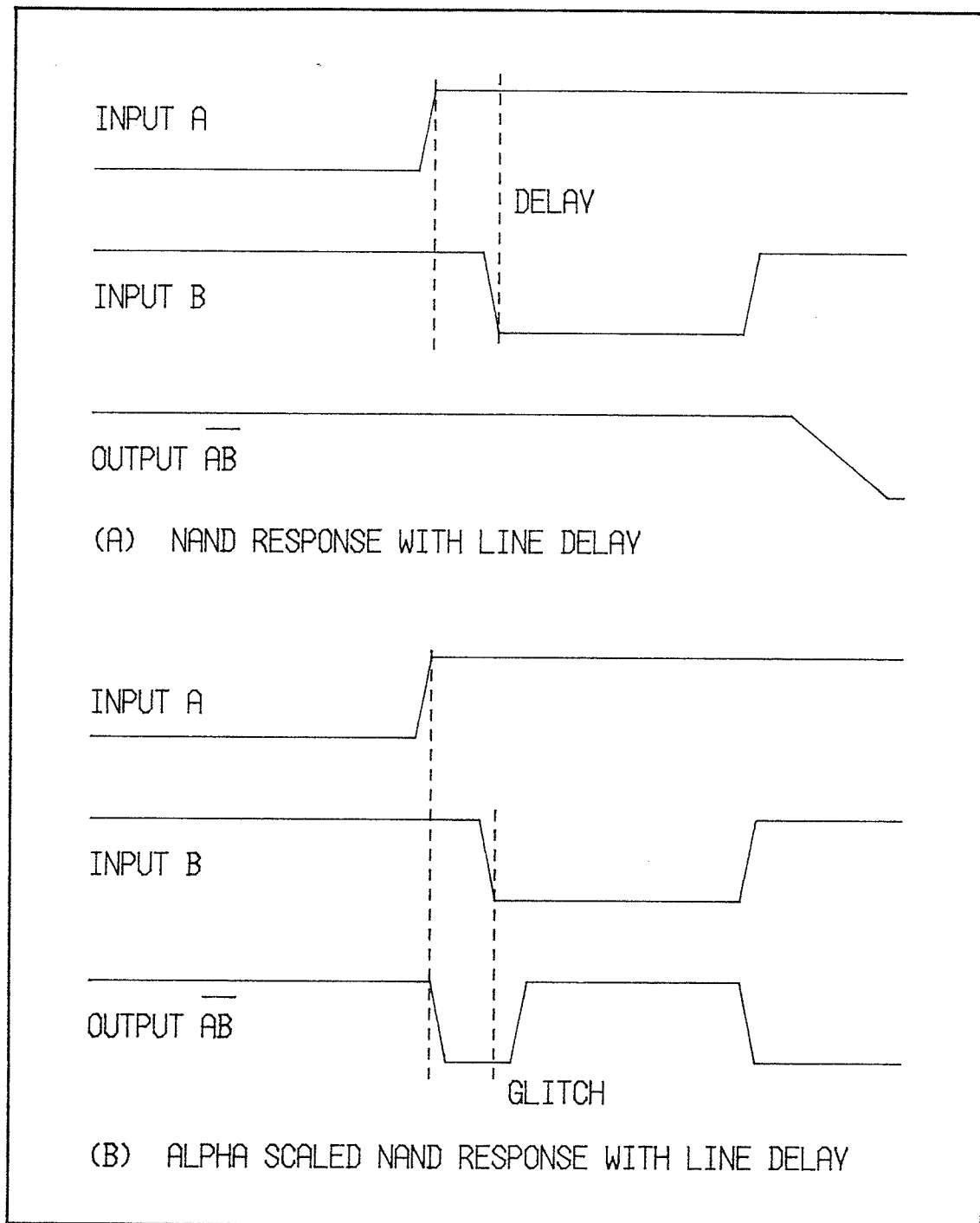


Figure 1: Alpha Scaling

and such solutions are likely to become much more difficult.

The problem of delays between synchronous elements has led to a number of alternate approaches at the system level. For example, communication between synchronous, locally clocked processors is provided by serial, asynchronous links in the InMosTM Transputer series. The assumption is that synchronizing many independent computational elements would be difficult and costly; hence, an asynchronous link is provided to facilitate interprocessor communication. Although the InMosTM Transputer is a microprocessor intended for parallel processing, the choice of an asynchronous communication link was made to eliminate the need for global clocks of any kind; hence, a large system could be relatively free from skew problems at a rack or system level. Additionally, the individual elements do not have to slow down to accommodate a structure like a shared synchronous bus. Although the Transputer is an architecture intended to solve the memory and communication problems inherent in parallel processing, many of the delay and system design problems encountered at the system level are increasingly experienced by logic designers. Some of the methods that are applicable at the board level may be appropriate to the very large scale integrated chips that will soon be in the design stages.

An alternative to a clocked, synchronous, inherently delay-sensitive system is to throw away the global, system-wide clock altogether. That is, to design with asynchronous systems

from the outset. Such methods have the property that system design and interconnection of unclocked elements may well be more simple, consistent and reliable. A correct interconnection of validated asynchronous elements would free the designer from the difficulty of accommodating the critical races and other delay dependencies of synchronous logic.

1.2 Self-Timed Systems

1.2.1 Definition of Self-Timed Systems

A self-timed system is any system that relies upon signal events embedded within data, rather than a global clock or control signal, to provide correct operation and synchronization [1]. It is a basic property that self-timed systems, or elements, are insensitive to arbitrary delays in their primary inputs and outputs.

1.2.2 Requirements for a Self-Timed System

In a self-timed system overall operation will be assured by a logical interconnection of elements that also exhibit a self-timed nature. There are a number of functional features that will allow us to construct logical devices that require

neither a global clock, nor careful interblock synchronization. The goal of self-timed design is to loosen or remove many of the synchronization requirements of current logic design methodologies.

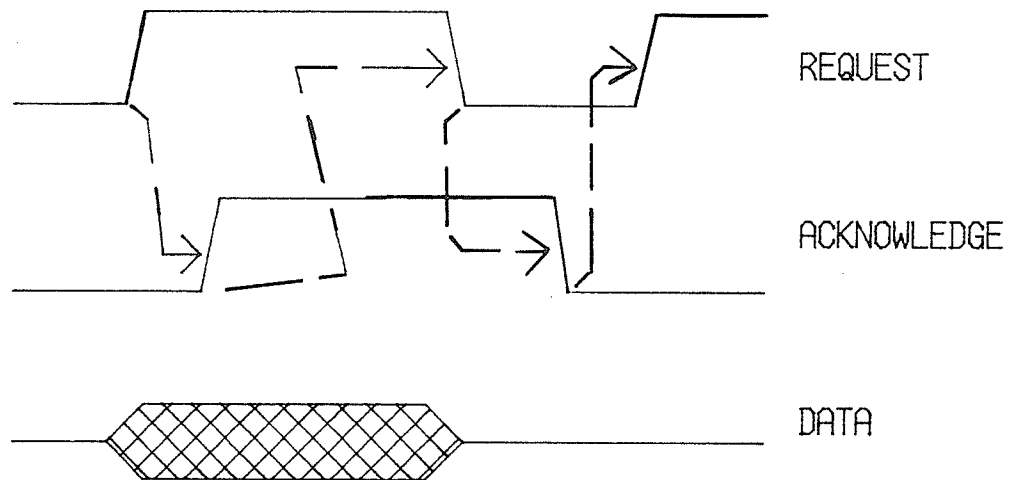
In order to provide clock-free realizations we must follow some basic rules:

- (1) Signal transitions must be unique, either from 0 to 1, or 1 to 0. Thus, an event is a transition without assumptions regarding the time delay inherent in any signal event.
- (2) Where order is not important to a design we define a partial ordering of delay insensitive lines. The design should assure that any reordering of events on these lines will not adversely affect a driven circuits outputs.
- (3) Logical 0 and 1 must be differentiated. Hence, a tri-state signalling of 0, 1 and null (-) is generally proposed. Optionally, distinct lines may be allocated to logical 1 or 0 to provide differentiation between the two. Data is represented by transitions from null to valid data and back again. This provides an inherent clock in the Self Timed Element (STE) data.
- (4) Synchronization is still necessary in some manner. A sending circuit which is forcing inputs into a receiving circuit must not place new data on the lines until the previous data has been processed.

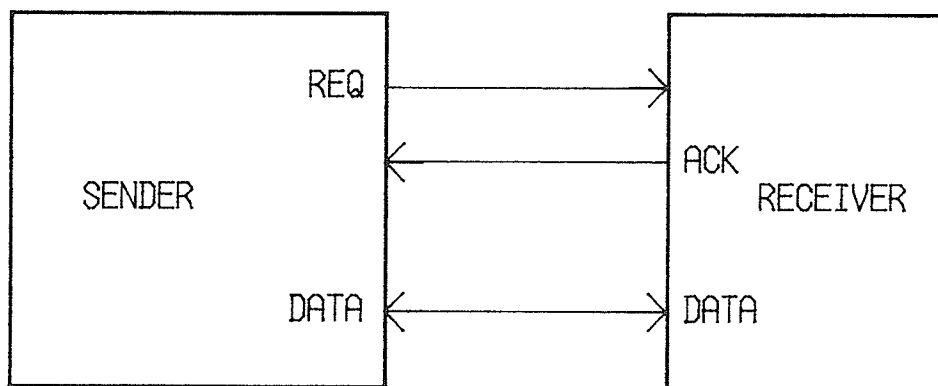
Hence, a handshaking protocol using request (REQ) and acknowledge (ACK) signals is typical. Protocol operation is relatively simple. The sender places data inputs to a circuit and a REQ onto the input lines of the receiving circuit. Once the receiver has carried out all appropriate processing tasks (logical or otherwise), the receiver sends an ACK signal to indicate readiness for more data. This is similar to physical layer handshaking on a network, or like that between a computer and a parallel printer. Refer to Figure 2 for an illustration of this process.

In order to provide design correctness, it is presumed that signal ordering may not be subject to circuit delays. Therefore, any signal ordering requires a distinct set of handshaking events for each set of ordered signals. Note that such a system does not, in general, preclude absolute maximums on delay. If limits must be placed upon delay, such as those necessary for dynamic logic, some relaxation of STE restrictions may be allowed. In such a case, some estimation of line delay may be necessary to assure that appropriate safety margins can be designed into the circuit.

The above conditions constrain circuit operation sufficiently so that the circuit can be said to be self-timed. Clocking is not derived externally but is inherent in the basic operation on the data. Such circuits tend to exhibit a well behaved nature at the block or system level; hence, the system



(A) HANDSHAKING PROTOCOL WAVEFORM



(B) BLOCK DIAGRAM

Figure 2: Handshaking

design task should be significantly easier.

1.3 Attributes of Self-Timed Systems

One of the major attributes of self-timed systems is a greater reliance on a hierarchical design methodology. A typical system, such as the one portrayed in Figure 3, would consist of several self-timed blocks interconnected in an appropriate fashion. Correct operation is based upon correct interconnection, not upon adequate delays and external sequencing/control. It is the nature of a correctly designed self-timed element, which are the functional blocks in the system, to inherently reject the need for time-critical sequencing. Thus, at the system level, design could potentially be clock free whatever the actual implementation details of the individual elements.

It is worthwhile to note that a STE does not have to be totally clock free in internal realization. A valid specification for a STE may include an external self-timed definition with synchronous, locally clocked logic implementing the function internally. Figure 4 illustrates just such a locally isochronic STE block.

If time sequencing is no longer a critical part of the design process, STEs promise to accelerate the design effort by significantly reducing the effort expended in the analysis of

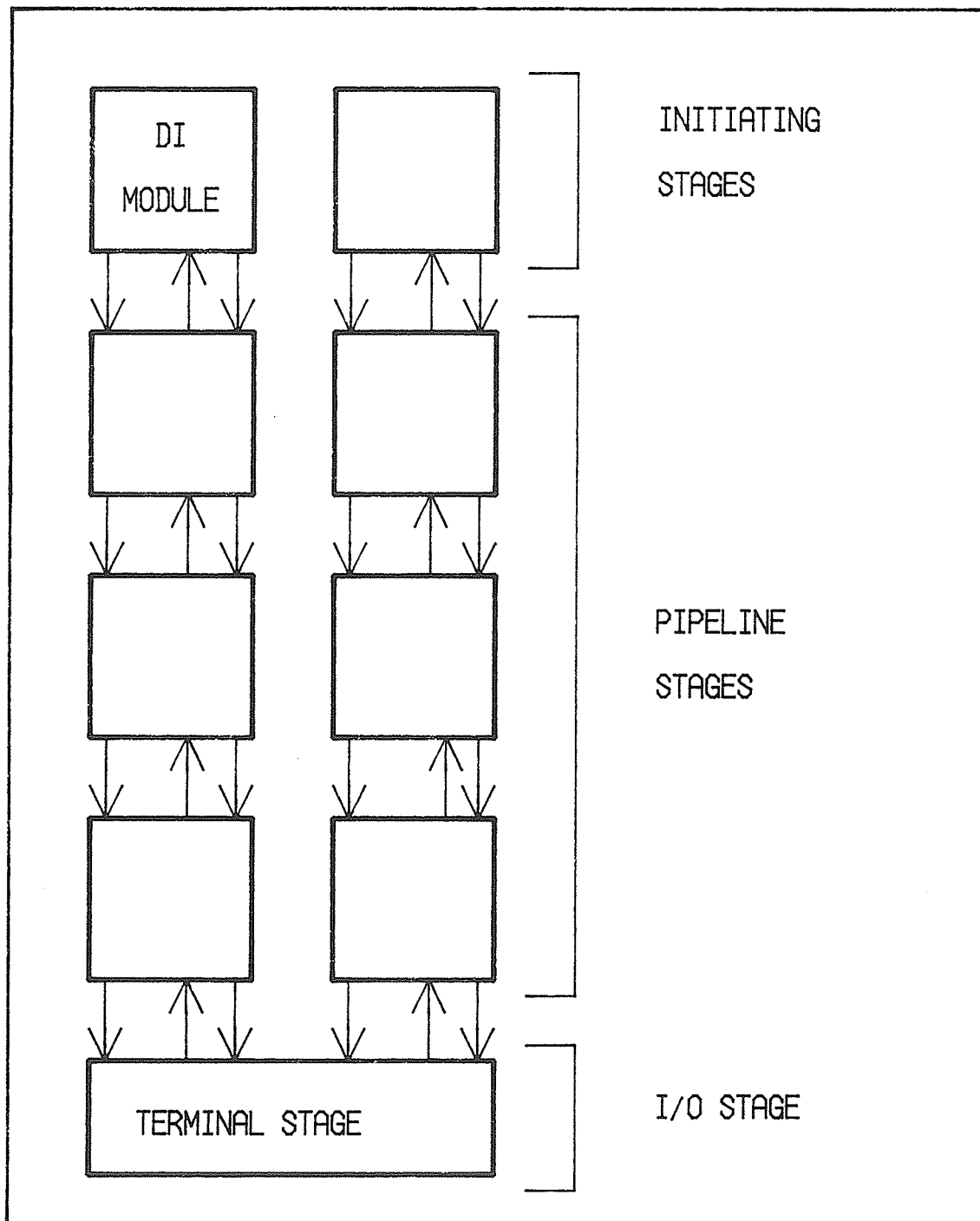


Figure 3: Self Timed Systems

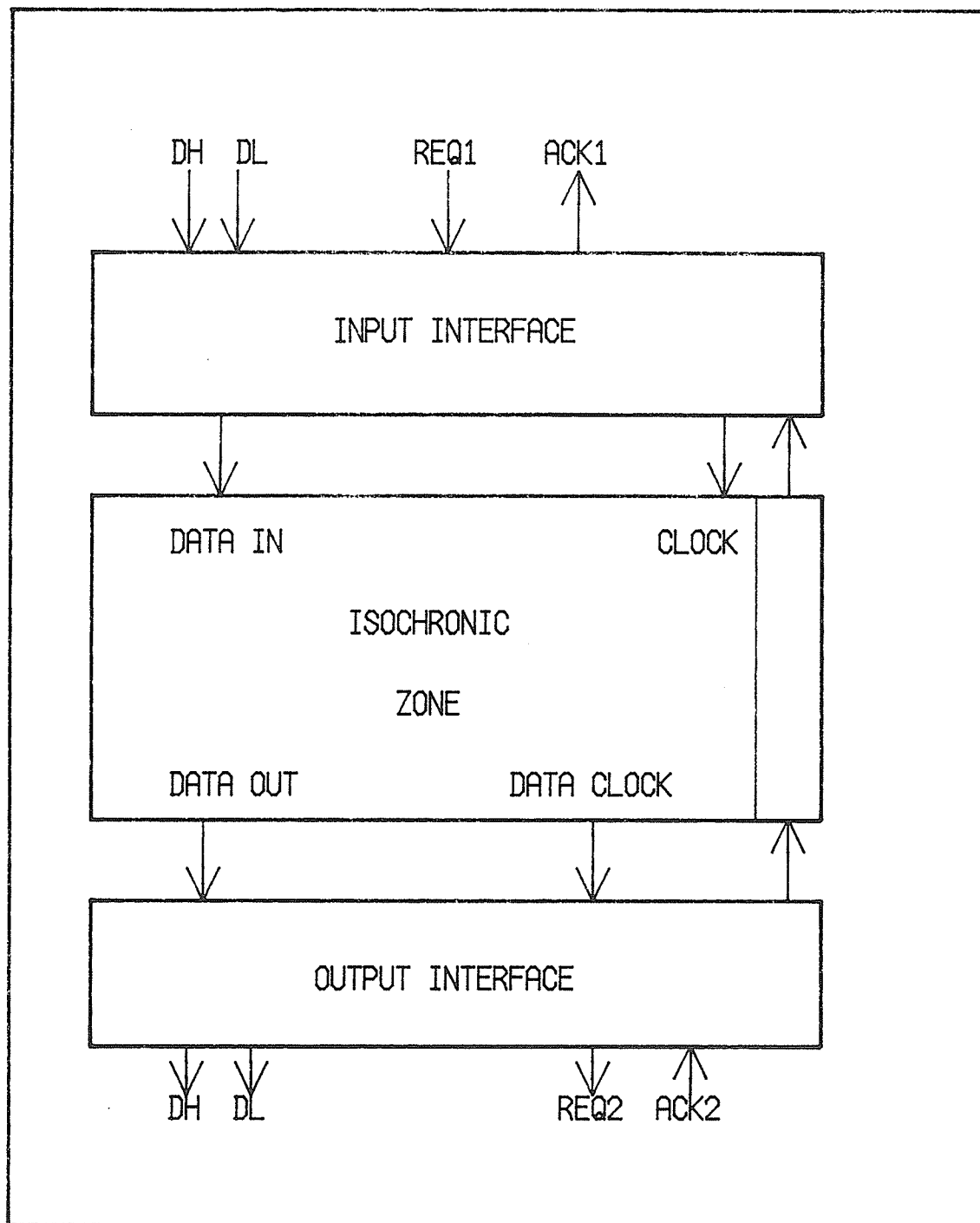


Figure 4: Isochronic Zones

clock distribution and sequencing in modern ICs and systems. Further, the STE approach should provide greater reliability in design since relatively little attention need be given to the line delays; only in the case of performance studies would line delay need to be extensively analyzed. This systematic approach is particularly attractive to designers who work at an IC standard cell level, or higher, since they will not be concerned with the design of the STEs themselves. The greatest current source of difficulty is the initial effort needed to produce a verified base of STE building blocks. In any case, STE design promises to ease the design process at the system building block level.

One reason that STE cell, IC, and board based methods appear so promising is that simple methods are available to provide pipelining, parallel synchronization, and cascading of such elements. Ullmann and others [2][1] mention simple ways to control ACK/REQ signals to produce such effects. Since blocks may be combined in sufficiently complex organizations, STEs may well be suited for design automation applications. A proper design is guaranteed by an appropriate interconnection of lines since delay is non-critical; hence, a tool need only provide the connections and a routing path. Thus, STEs provide significant support to some of the design methods and tools that will be needed for the dense Ultra Large Scale ICs (ULSI) of the future.

A number of authors, such as Molnar [3] and Fang [2],

provide a class of STEs which are even more flexible. Their class of circuits, called Delay Insensitive (DI) modules consist of input and output signals whose time characteristics are only partially ordered. That is, a request and a proper subset of inputs have switching events prior to each ACK signal. In this class of circuits, Delay Insensitivity permits any arbitrary delay to be inserted in any line while guaranteeing correct circuit operation. DI circuits generally differ from STEs only in absolute adherence to DI limits and incorporate a general requirement of resistance to metastability. As well, it is uncommon for a DI network to use a null state to differentiate data bits. Instead, all clocking is incorporated into the data lines without the use of a null signal. If a DI circuit requires ordering, it is necessary to provide separate ACK signals for each and every ordered event. The most general form of STE system is, therefore, a DI system. The next chapter will detail many of the attributes of DI design.

CHAPTER 2: DELAY INSENSITIVE SYSTEMS

2.1 Delay Insensitive Modules

Delay Insensitive (DI) modules are logic generating circuits that satisfy the delay insensitivity condition. A DI module is said to be in an environment which introduces a finite but unknown delay in input and output signals. Figure 5 illustrates the environmental to modular relationship in DI systems. Thus, given an environment with finite delay, a signal set $\{A_1, A_2, \dots, A_n\}$ could, even if the arrival order at the environment boundary is known, be received in any arbitrary reordering at the circuit boundary $\{a_1, a_2, \dots, a_n\}$. If the output of the circuit, $b_k = f_k\{a_1, a_2, \dots, a_n\}$, is produced with consistent correctness regardless of signal reordering, then we may indicate that the element is Delay Insensitive.

Delay Insensitive modules conform to the basic rules of STEs with a number of extensions. The fundamental rule set is as follows:

- (1) All inputs are semi-modular. That is, the only legal transition on any line is from logical 1 to logical 0, or vice-versa, during any one full cycle of

request/acknowledge. Further, there must be a monotonic and stable transition; examples of non-monotonic transitions would be glitches, or other transients, on the circuit inputs.

(2) A circuit cycle is initiated by a partial ordering of signals and REQ, and is terminated, after an arbitrary internal delay, by circuit output and ACK. That is, some signals may be asserted with REQ in any order, followed by assertion of valid outputs and the ACK signal. Proper design consists of specifying which combinations of signals may occur.

Practical circuits will also have logical 1 or logical 0 data encoded in a special manner; either separate lines are usually employed for 1 and 0, or a tristate signalling method must be utilized. Practical design assumes a two line configuration with the general goal of eliminating all explicit circuit clocking in DI realizations. If a module satisfies the above two rules, it is said to be Delay Insensitive.

2.1 System Considerations in DI Modules

2.1.1 Signal Conventions in DI Modules

Signals in DI modules are said to be semi-modular and monotonic. Once a transition or excitation on a line is

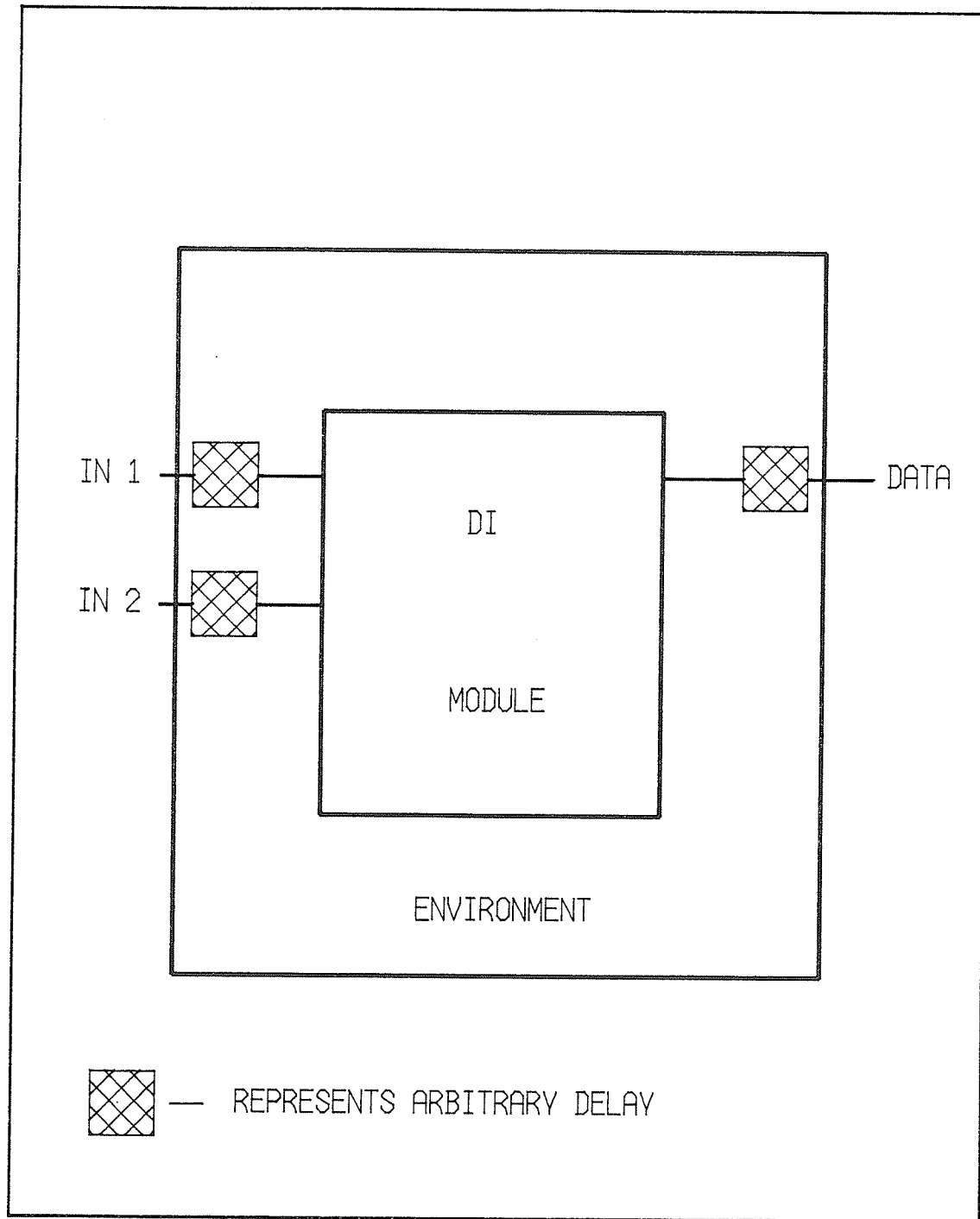


Figure 5: DI Modules

initiated, it must progress to the other line state without multiple transitions. The only way an excited signal returns to stability is by entering a complementary state. We refer to this behavior mathematically by writing the expression $0 \rightarrow 0^* \rightarrow 1 \rightarrow 1^* \rightarrow 0 \dots$ where the asterisk represents the excitation or instability condition.

Fundamentally, the semi-modularity property is critical to forming a DI system. The implication is that any transition on an input line may occur only once in the circuit REQ/ACK cycle. If we allow multiple transitions on a line in each cycle, we are then assuming time dependencies on the part of the DI module. This would violate delay insensitivity since we now must know the internal and environmental performance of the system. Therefore, it is typical that some subset of signals will undergo transition along with the REQ signal. Further variation of input lines is thereafter forbidden until an appropriate ACK signal reaches the sending circuit. Figure 6 illustrates the behavior of a system having a partial ordering of inputs. All limitations upon signal transition are entirely dependent upon the functional specification of the DI module; hence, partial ordering is a design matter that depends upon the specific application for the DI module.

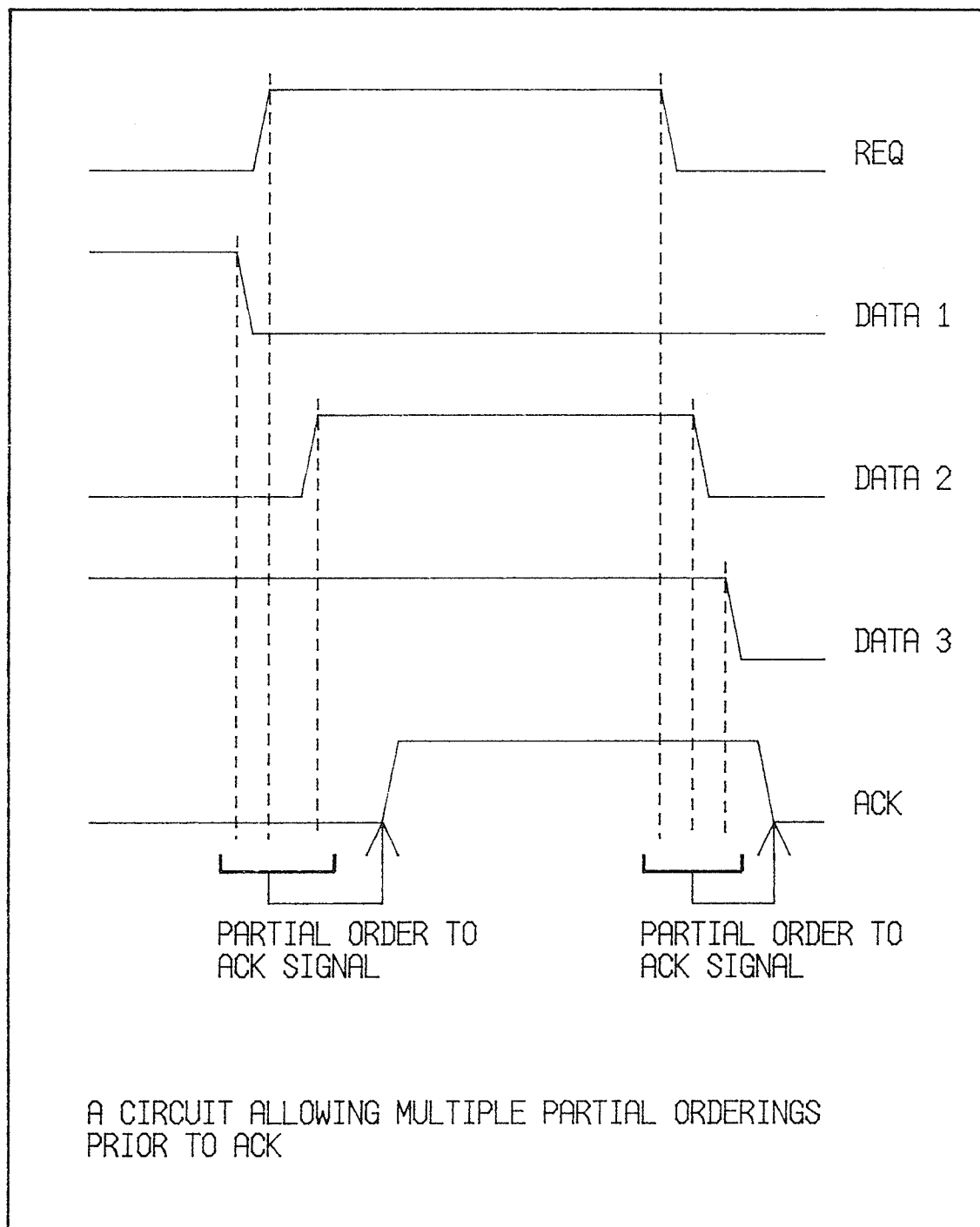
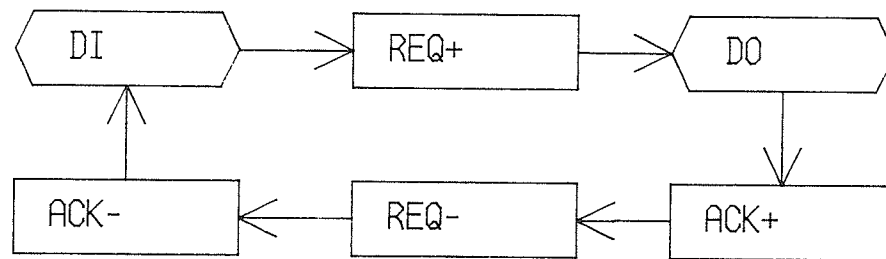


Figure 6: DI Partial Ordering

2.1.2 Synchronization Signals

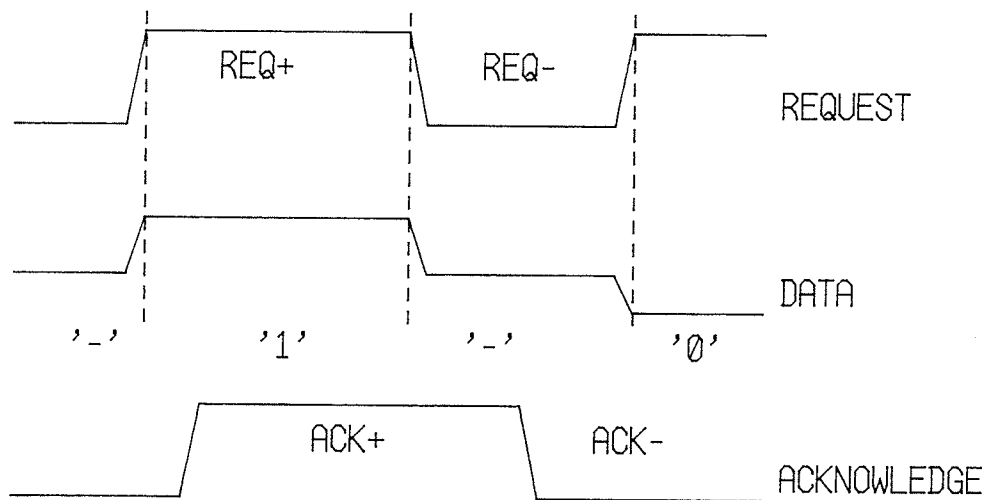
In all self-timed systems, such as a DI module, it is necessary to have a signal that initiates a cycle or functional state change in that module. In DI systems, this function is served by a simple handshaking protocol. A protocol of REQ followed by ACK can take a number of forms, the simplest of which is called the single rail, or 4 cycle, protocol. In all such handshake protocols the basic cycle is of data input and REQ, followed by data output (state change) and ACK. Figure 7 (a) illustrates the single rail protocol in state terms, while Figure 7 (b) provides a depiction of typical waveforms that would satisfy the single rail protocol. In this type of protocol the signal must return to a null level (commonly 0), hereafter referred to as REQ^- and ACK^- , prior to beginning a new cycle. In practical applications it is convenient to encode REQ^+ and ACK^+ as a logical 1, and REQ^- and ACK^- as a logical 0. Please note that 0 and 1 can easily be interchanged if done so in a consistent manner. In this thesis I assume that the logical 1 and 0 values will correspond to the standard voltage levels for CMOS and TTL.

Although this is not the only formulation for an STE protocol, it does serve as a basic starting point for module construction. We will find that the use of protocols and a fundamental circuit called the "join" will allow us to form more complex circuits.



DI - DATA IN
DO - DATA OUT

(a) Four Cycle Protocol



(b) Four Cycle Waveforms

Figure 7: 4 Cycle Protocol

2.1.3 The Muller C Element (Join)

One circuit that can be shown to be fundamentally self-timed and delay insensitive is the Muller C element. Although it does not have a REQ/ACK protocol, Miller [4] demonstrates that it satisfies all the criteria of a self-timed block. Further, it is inherently delay insensitive and satisfies all DI conditions as shown by Molnar [3], and others [1][4][5]. This circuit is often called a join function because of the unique properties it exhibits.

As a two input circuit, the join will produce a 1 output if all its inputs are a 1, or a 0 if all its inputs are a 0; otherwise, it retains the previous output state. Table I shows the logic function of the Muller C in a truth table format. Delay variation cannot adversely affect this circuit other than to cause delay variation at the outputs.

TABLE I: MULLER C ELEMENT TRUTH TABLE

| A | B | OUTPUT |
|---|---|----------|
| 0 | 0 | 0 |
| 0 | 1 | OUT(t-1) |
| 1 | 0 | OUT(t-1) |
| 1 | 1 | 1 |

Why is this circuit important? This circuit's importance results from the ease with which DI modules and STEs are cascable to form a variety of configurations from pipelined to parallel operation. The Muller C serves to provide the global synchronization that would otherwise be provided by a clock in many systems. Further, if it is desired that action

be synchronized upon a specific set of signals, a simple cascade of C elements is known to work consistently. In a circuit sense, the Muller C function may be freely reorganized or extended in the same way a boolean OR or boolean AND function may; that is, it is fully associative. Figure 8 illustrates the action of "joining" signals to provide a synchronization event. In this case, we are joining several ACK signals to provide some system level synchronization in the form of a common ACK. In practical realizations, the join is most commonly used to provide synchronization of control signals in pipelined, or parallel, final stages.

2.1.4 Module Forms

The fundamental DI module has data inputs, data outputs, a REQ, and an ACK signal. Although there are ways to construct DI systems without an explicit REQ/ACK, there must be a mechanism of feedback present in the system; otherwise the system cannot possibly provide delay insensitivity. Typically such signals are present explicitly. A typical central module would appear as shown in Figure 9 (a). The number of inputs and outputs in a DI module is arbitrary and fully dependent upon the specific implementation. The basic module format depicted is, in itself, quite useful. This form, with one or more inputs, is often used as a basic building block for more

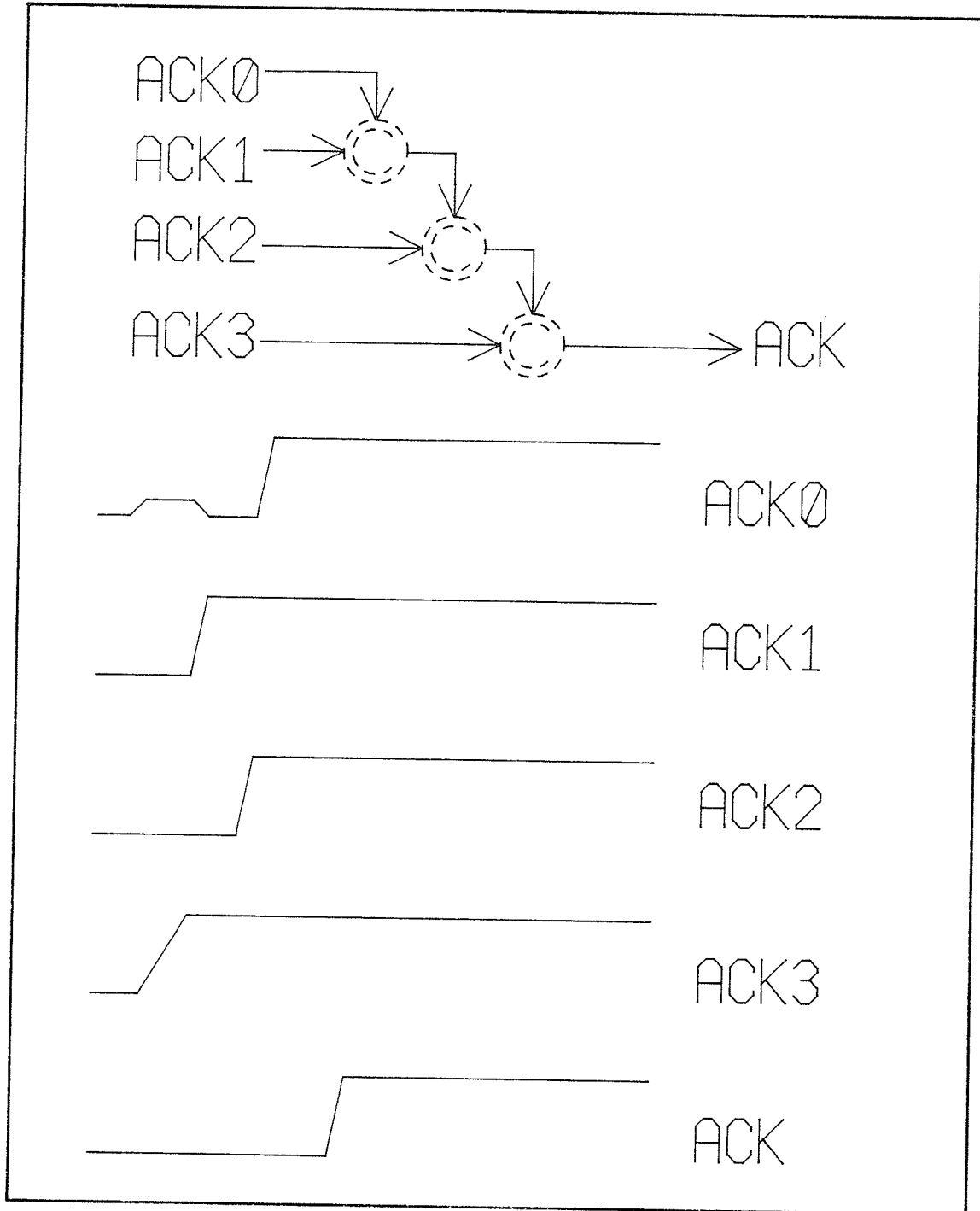
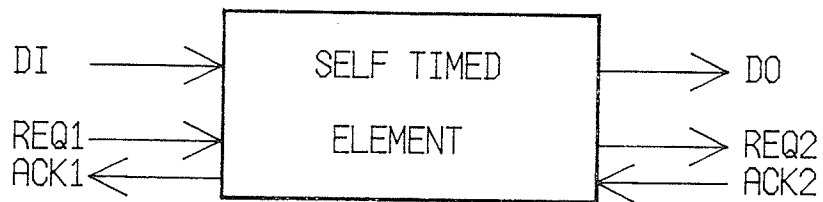
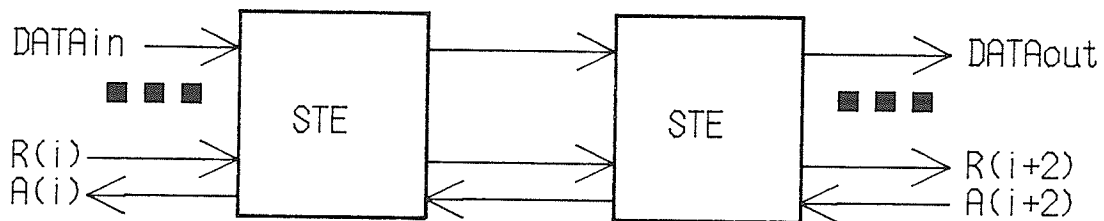


Figure 8: Multiple Joins



(A) - General Self Timed Module

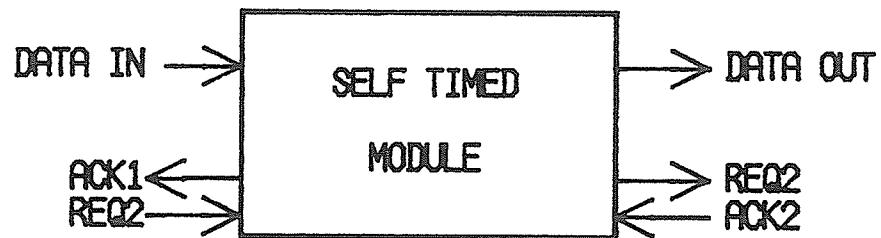


(B) - Pipelined Self Timed Elements

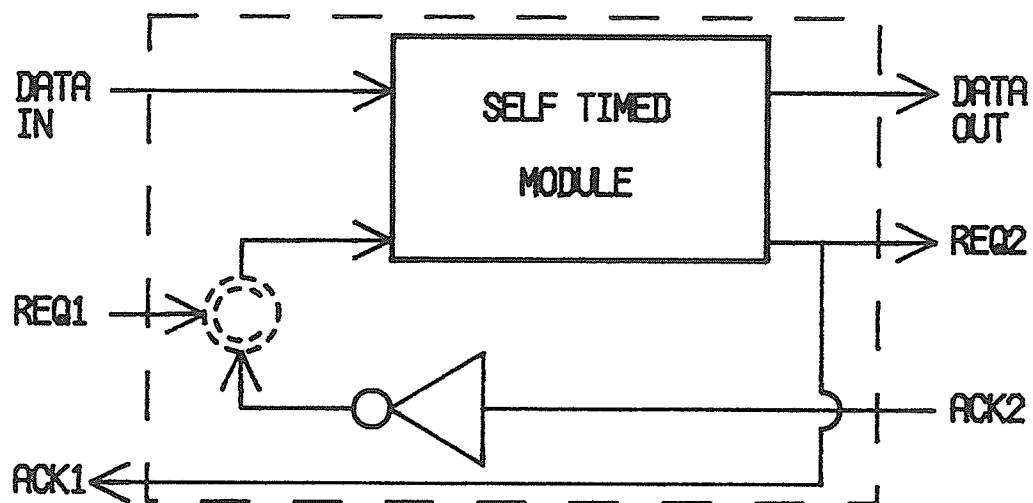
Figure 9: Pipelined STEs

complex circuits.

For example, it is common to desire some level of pipelining in a circuit design. Figure 9 (b) illustrates how STEs could easily be pipelined. In a pipeline system, changes must propagate up the full chain in a lockstep fashion before we dare begin a new cycle at the base of the pipeline. For example, a FIFO could be envisioned as a chain of DI registers connected in series. It is convenient to note that we do not have to adopt a special design to cope with pipelining. All that is needed is a standard module and a Muller C element. Figure 10 shows how a DI module could be prepared for pipelined operation. Assuming the system started at a known state (usually 0), an incident REQ would initiate module activity. The module would then generate an ACK signal as appropriate; this ACK serves as both the outgoing REQ to the next module and the returned ACK to the sender. Is this a contradictory condition? No, it is not; the outgoing REQ initiates the next pipeline stage, while the ACK will allow completion of the current cycle at the sender. If the sender starts a new cycle, it will have to wait until the module receives an ACK from the pipeline stage downstream since REQ1 and ACK2 are joined by a Muller C function. Thus, the pipeline does proceed in lockstep. In any signal ordering, new transitions into the module must wait for both a valid REQ1 and ACK2. Therefore, the next pipeline stage must be ready for more activity before further operations can continue. The C element assures correct



(A) PIPELINED SELF TIMED MODULE



(B) PIPELINED MODULE FROM A BASIC SELF TIMED MODULE

Figure 10: Pipeline Modules¹

operation by functional design.

Similarly, modules can readily be connected in parallel. All that must be done is to connect a series of ACK signals through an appropriate number of Muller C elements. Figure 11 illustrates just such an organization. A common ACK signal is generated by the circuit and is fed back to all the individual parallel module inputs. As before, the join will facilitate the synchronization activity at this point. By utilizing appropriate combinations of STEs and Muller C elements major architectural variations are easily accommodated.

2.2 Delay Insensitive Protocols

One useful methodology in circuit design is to use a delay insensitive protocol between major blocks of circuitry. This protocol may incorporate either a parallel or serial communication path complemented by the appropriate use of a REQ/ACK signal set. By using a standard protocol to connect major functional blocks, the block design process may be simplified to incorporate standard components/elements. Further, a standard protocol between blocks would simplify the system integration process. The following paragraphs will discuss a tristate protocol system as one method for block interconnection.

In the tristate or pseudo-tristate method, all data must be formatted so that there are no time dependencies on signalling

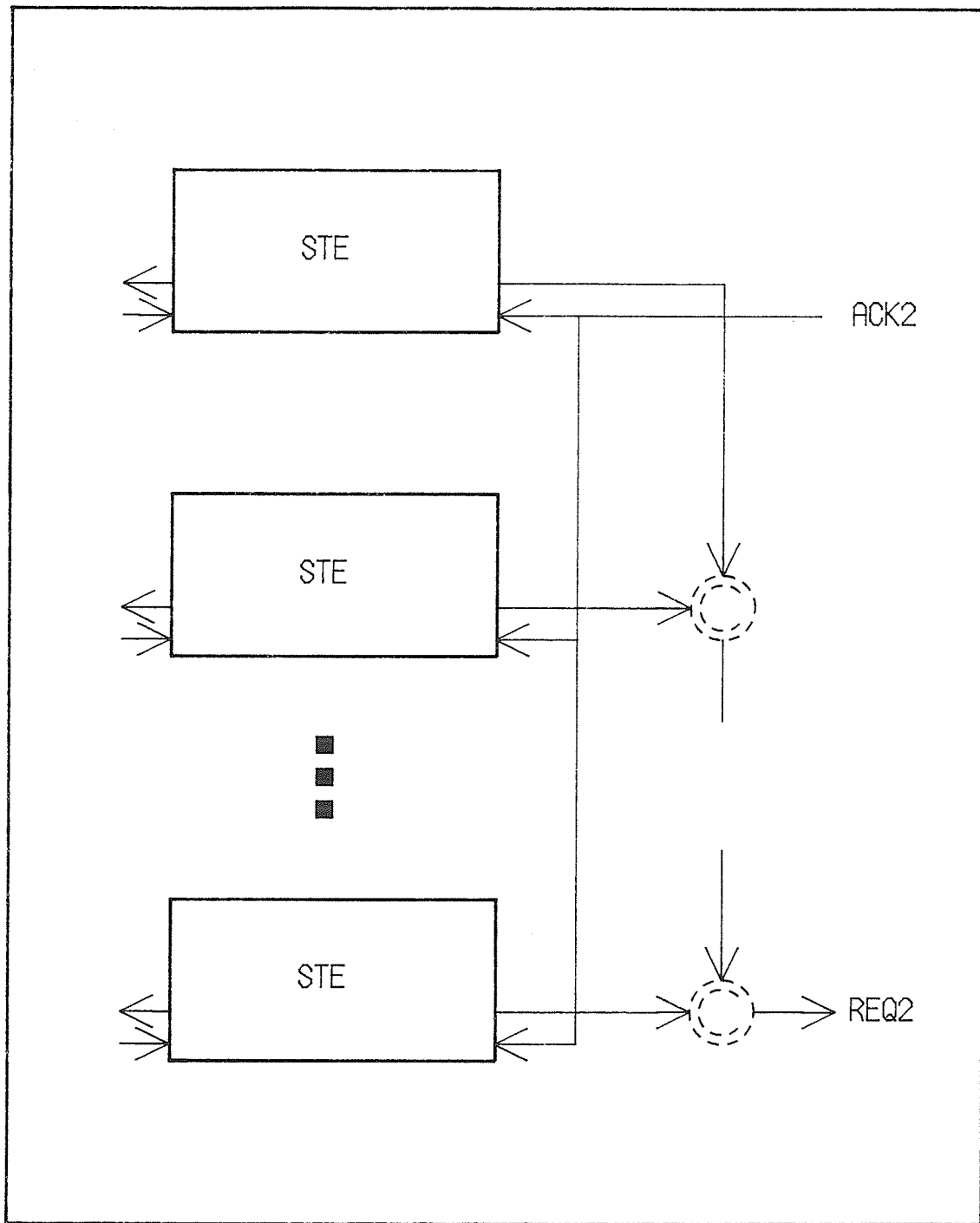


Figure 11: Parallel STEs

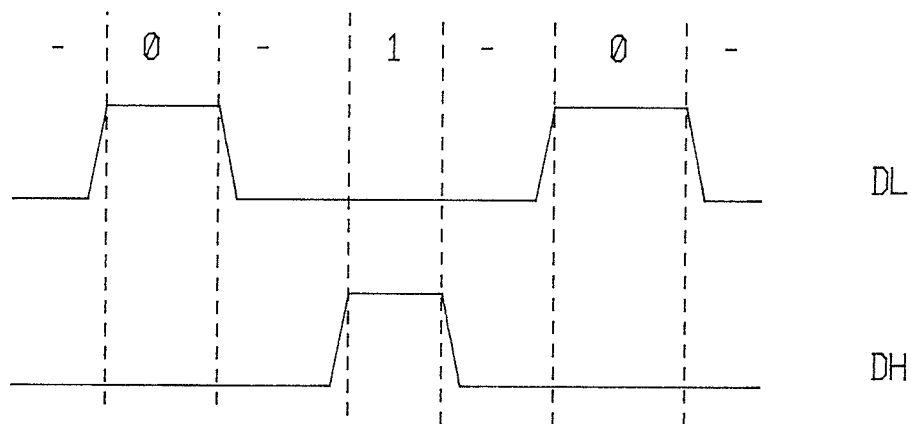
a logical 1 or 0 to the receiving circuit. A common method is to provide two lines for each logical data bit since true three state data lines are quite difficult to fabricate in practical circuits. Two such lines generally encode a '1', '0', and null ('-') state. Data lines must return to the null state between each and every transmitted data bit, much as a return to zero code does. Typically, one line is labelled as a high data line (DH) and the other a low data line (DL). Figure 12 shows a sample encoding scheme, and sample waveform, that implements a pseudo-tristate. Each transition from null to a valid data value, and from valid data to null, will require one full REQ/ACK cycle. The implication of such a signalling method is that there will be significant protocol overhead. An alternate method involving single line transitions without a null state has great advantages over the tristate line.

The REQ/ACK signal pair may have one of two convenient signal conventions: (1) single rail. (2) double rail. In the single rail convention, both REQ and ACK must return to a null state prior to initiating a new cycle. Thus, as Figure 13 illustrates, the handshaking signals return to '0' as triggered by signal edges. The single rail protocol is the simplest of the two protocols to implement in practical logic.

Unfortunately, a single rail protocol has a significant weakness. The total data cycle requires that REQ and valid data travel to the receiver, followed by the return of ACK to the sender. Next the sender must null the data lines using the

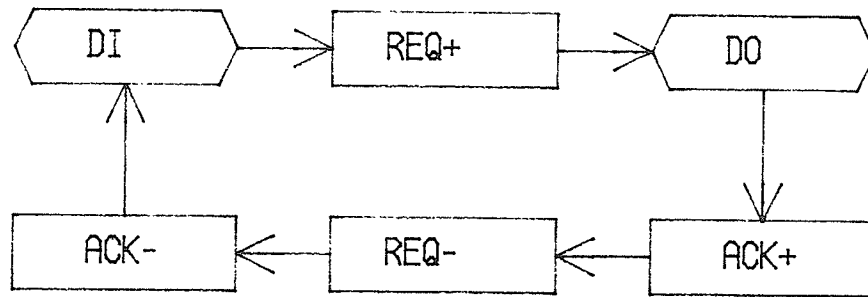
| DH | DL | |
|----|----|-----------------|
| 0 | 0 | - NULL |
| 0 | 1 | - LOGICAL 0 |
| 1 | 0 | - LOGICAL 1 |
| 1 | 1 | - UNDEFINED (-) |

(A) LOGIC TABLE DEFINING A TRI-STATE LINE



(B) WAVEFORM FOR A TRI-STATE LINE

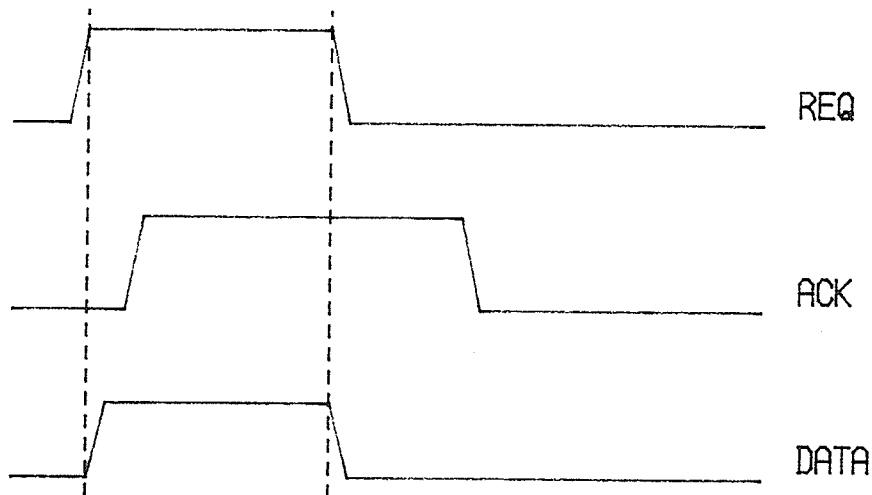
Figure 12: Tri-State Line



DI - DATA INPUT
DO - DATA OUTPUT

REQ - REQUEST
ACK - ACKNOWLEDGE

(A) STATE DIAGRAM OF THE SINGLE RAIL PROTOCOL



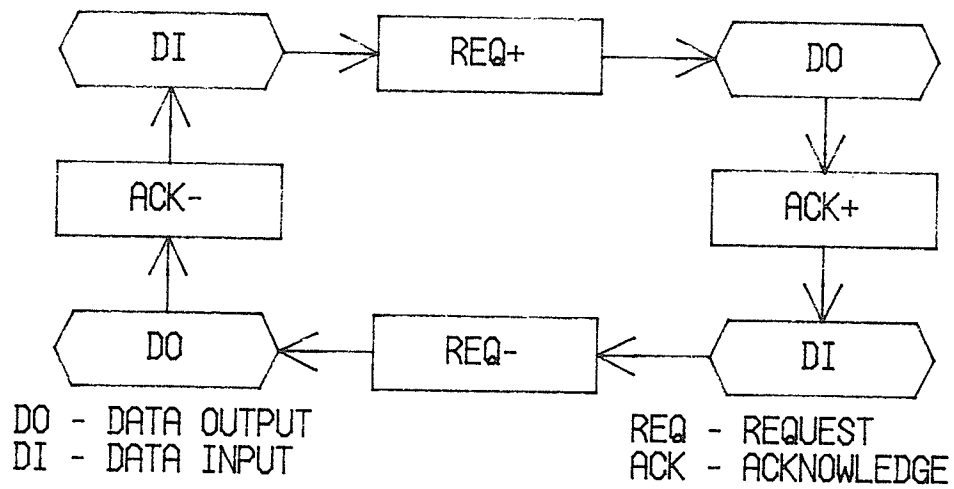
(B) SINGLE RAIL PROTOCOL WAVEFORMS

Figure 13: Protocol (1 Rail)

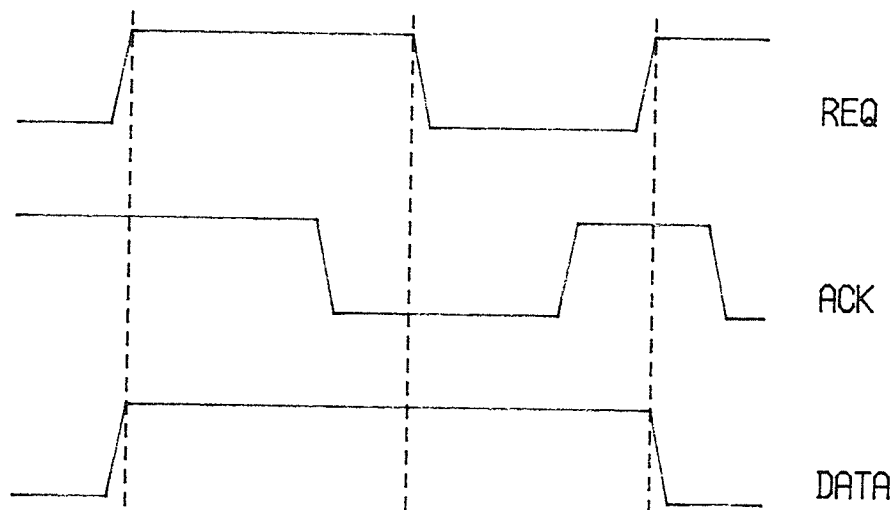
same REQ/ACK cycle sequence. Over a long trace having significant delay there could prove to be significant overhead. A double rail protocol remedies this problem. The double rail protocol relies upon a single, unidirectional transition to signal an ACK or REQ. Figure 14 shows a protocol signal set where each pair of transitions on the REQ/ACK lines signals is a full data cycle. In this protocol the REQ/ACK pair must still travel between the sending and receiving circuit; however, this happens only once each way, not twice as in the single rail protocol. Since the protocol lacks a null state, it is much more efficient. The only drawback to a double rail protocol is the reality that circuit implementations are generally significantly more complex, and hence, significantly larger in circuit area than a single rail implementation. Figure 15 illustrates the edge encoded nature of data in a double rail DI protocol. Although the side effect of using a double rail is the probability that protocol handling will consume more IC area, the significant advantages in performance may well outweigh any sacrifice in silicon.

2.3 Design Considerations in DI Systems

When producing a design using DI principles and modules, it is necessary to assure a number of basic conditions. First, design "correctness" by interconnection is critical. If the



(A) DOUBLE RAIL PROTOCOL STATE DIAGRAM

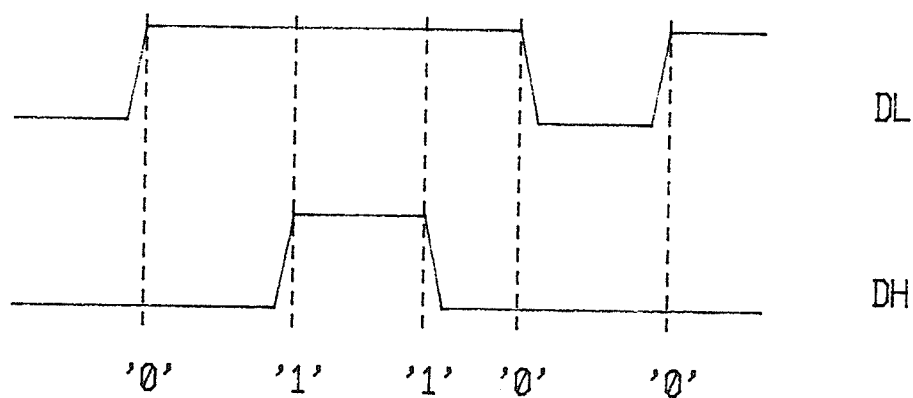


(B) SINGLE RAIL PROTOCOL WAVEFORMS

Figure 14: Protocol (2 Rail)

| DL | DH | |
|------------------------------------|------------------------------------|-----------|
| $\overline{\downarrow} / \uparrow$ | $1 / 0$ | LOGICAL 0 |
| $1 / 0$ | $\overline{\downarrow} / \uparrow$ | LOGICAL 1 |

(A) DOUBLE RAIL DATA PROTOCOL TABLE



(B) DOUBLE RAIL DATA WAVEFORM

Figure 15: Double Rail Data

modules are not related by a general protocol, then the design blocks must mesh very tightly in order to provide adequate or correct operation. Simplifying system level integration is the essential reason for using protocols in the first place. Second, a sense of "liveness" must be assured. Liveness refers to having a network free from deadlock. Deadlock, or deadly embrace, is a situation in which two or more modules are waiting for a signal from each other in a configuration that cannot be satisfied, thus halting operation; functionally, they are all waiting upon each other. This is a problem peculiar to DI and asynchronous networks in general since continuation, completion, REQ/ACK, and other signals used for circuit operation and arbitration operate in a lockstep fashion. Great care must be taken to avoid such classes of problems in DI systems.

When implementing DI modules, it is fundamental that the circuit be a critical-race free, non-fundamental mode circuit. DI modules must accommodate signal re-ordering; thus, a circuit must be tolerant of switching orders that include the possibility of simultaneous switching. Since many circuits could be produced with state machines, care must be taken to permit any valid reordering of signals in circuit switching behavior.

The internal organization of a DI module is arbitrary and wholly dependent upon the designer. If desired, a fully unclocked design may be produced. Unfortunately, it is often

much more difficult to produce a design having no clocks. An alternative is to rely upon a local clock within a limited region within the IC. Such local clocks are not synchronized globally. A clock operating in a limited, synchronous region is called an Isochronic clock. The region that it operates within is called an Isochronic Zone. Figure 16 contrasts Isochronic and fully unclocked DI modules. Building circuits that are locally synchronous has the advantage that synchronous design over small areas is well understood, and quite reliable since line delay is relatively limited. Therefore, an IC could be globally DI while it has regions of synchronous logic within its Isochronic zones. Any design method within those zones is valid as long as it satisfies DI behavior in a global sense. If communication into and out of the zone is DI, than the device is said to be DI at a system level. Chapter 4 will present a general method for DI design.

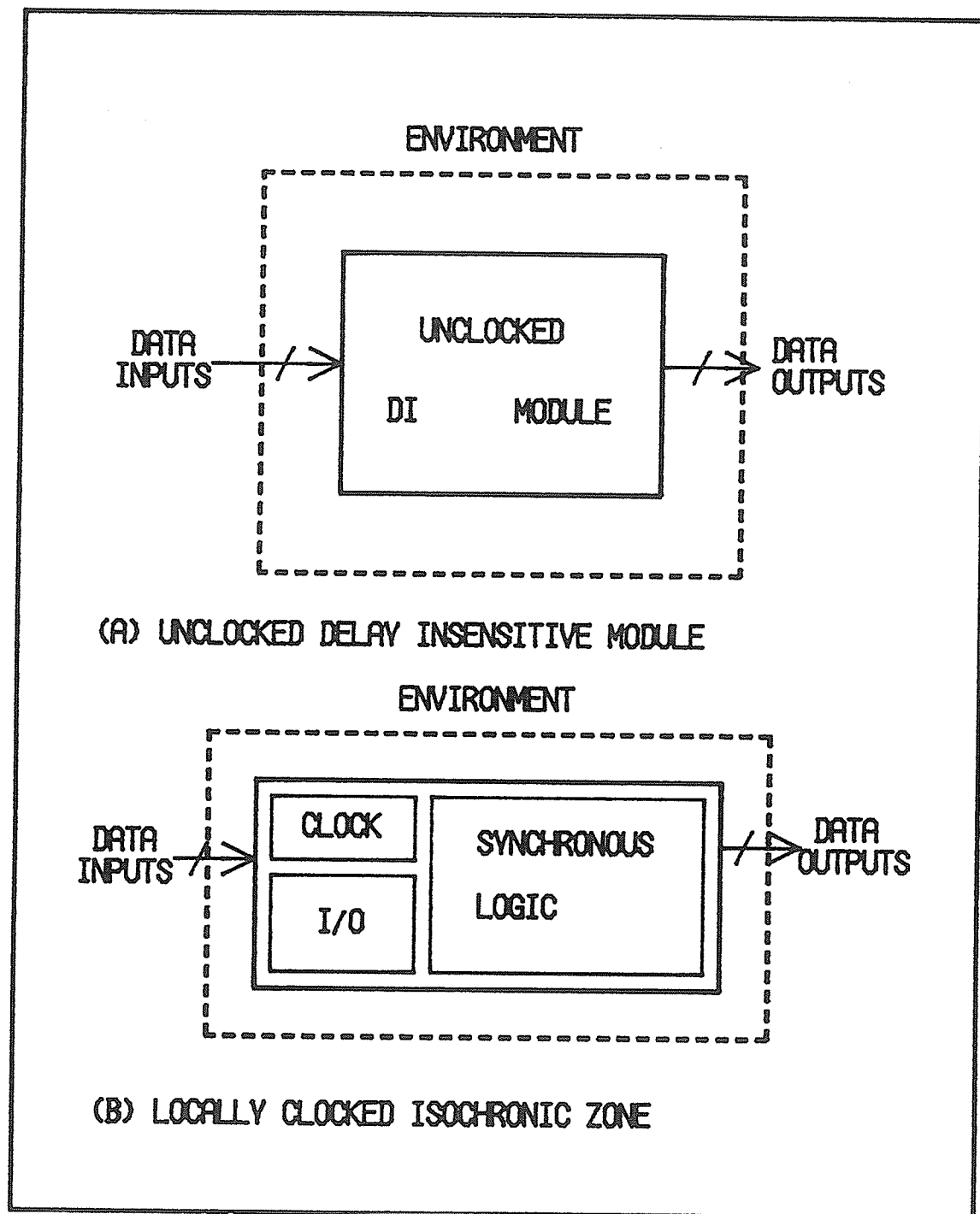


Figure 16: DI Modules

CHAPTER 3: AN ASYNCHRONOUS DI INTERFACE IC

3.1 Implementation Goals

A major goal of this thesis is to implement a DI Protocol in a practical CMOS IC. The IC is to implement a number of serial channels for byte-sized, asynchronous communication between microprocessor and microprocessor based devices. Further, the IC is to incorporate only local clocking methods for all fundamental circuit operations within the serial channels. The device uses the fundamental asynchronous DI protocol signals to derive all signals within the channel's single isochronic zone. This allows us to use synchronous logic elements, such as D flip-flops from the University of Manitoba VLSI cell library, while still retaining many of the benefits of avoiding global clocking. Essentially this IC is implemented as a set of locally synchronous internal blocks that are linked to a DI, asynchronous protocol.

The protocol that each channel provides is a single rail data protocol coupled with a double rail REQ/ACK protocol. Selection of this combination was made for the following reasons: (1) Single rail data signalling provides much

simpler clocking while supplying framing information for the data words. (2) Double rail REQ/ACK signalling provides significantly greater performance while adding only a moderate increase in circuit complexity. (3) The mix of double rail REQ/ACK and single rail data proved to be convenient for deriving appropriate clocking for the channel's internal synchronous logic.

Appropriate interface logic is provided to allow the DI channels to be linked to a typical synchronous processor. The Asynchronous DI Channel (ADIC) IC was designed to behave as a memory mapped, interrupt driven device; therefore, control is via memory mapped registers.

This design demonstrates that synchronous and asynchronous design elements may be successfully mixed. Further, it illustrates the advantages of a DI protocol approach at a system level.

3.2 Protocol Specifications

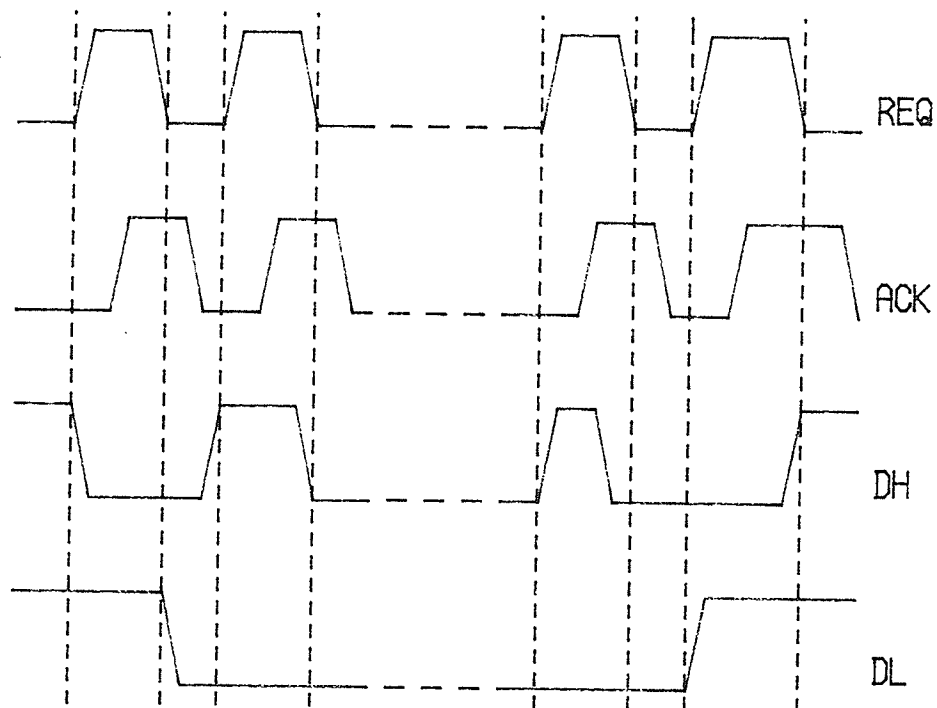
As has been mentioned, the IC implements a number of serial channels for asynchronous communications. Since this chip provides communication between current microprocessor based systems, it transmits information in a standard byte format. In this manner, software using current serial chips would need to be minimally rewritten. Only details of control and

initialization would vary, not the data formats. Thus, line signalling and waveform activity would be transparent to the programmer.

The IC protocol is DI and asynchronous. Figure 17 demonstrates a double rail REQ/ACK protocol with one single rail data channel. There is one simple variation on the fundamental single rail data protocol in this IC. Not only are logical one, zero and null provided on the data lines, but a data framing signal is also encoded. A sequence of DH DL equal to 11₂ indicates that the line is between data bytes. Therefore, this data value can serve to frame the data into bytes and provide channel initialization. In fact, the implementation forces a reset of certain internal states, as appropriate, upon receiving a framing value on DH and DL.

Please note that both the placement of the data event, and the null event, upon DH DL each require a full REQ/ACK sequence. Such behavior is necessary because the sender could otherwise be required to incorporate time metrics in its transmission behavior. It is desired that assumptions about delay, in either the interconnecting lines, or the receiver's internal behavior, be incorporated into the design of the system. Examination of Figure 17 clearly shows this relationship.

The DI protocol selected provides a number of major advantages over typical microprocessor serial channels. First, the hardware designer does not have to presume a specific



(A) PROTOCOL WAVEFORM FOR 1 BYTE

| DL | DH | VALUE |
|----|----|-------------------|
| 0 | 0 | NULL (-) |
| 0 | 1 | LOGICAL 1 |
| 1 | 0 | LOGICAL 0 |
| 1 | 1 | DATA BYTE FRAMING |

(B) PROTOCOL FRAMING CONVENTION

Figure 17: Chip Protocol

transmission rate for the hardware. The channel will operate as quickly as internal delays and the communication medium will allow. Thus, if coaxial cable connects the two channels, operation will be automatically raised to the limits of the medium limits by the protocol itself. Since time metrics are not assumed, the protocol operates as quickly as the medium will permit the REQ/ACK cycle to operate. Systems connected in this manner would have a wide latitude in interconnect variation and upgrading. Further, low level synchronization is inherent in the protocol itself; thus, initialization of a network is simplified. Although some software protocol synchronization is needed on all networks, this task should be somewhat simplified under this type of organization. Unfortunately, a DI protocol does extract some cost from the system--in this case time. The REQ/ACK handshaking that is inherent in a protocol can double the best case time in many serial channels. Further, a tri-state single rail code reduces performance further by requiring that both data and null have a REQ/ACK cycle. Potentially, such a channel's maximum speed could be as little as one quarter of the serial channel maximum. On the positive side, the performance of this protocol is based upon edges rather than levels; hence, overall behavior should be quite acceptable in many systems. Additionally, it is quite difficult to provide reliable asynchronous communications in many systems. A DI protocol, such as the one above, could regularize many design tasks in

such computer networks/systems.

3.3 Physical Implementation

The ADIC IC was implemented using three micron Northern Telecom CMOS. Static design was utilized extensively throughout the IC since estimates showed that pin-limitation, not silicon area, would determine the number of channels that could be placed on-chip. The following sections will detail the ADIC IC as implemented.

3.3.1 Input Serial Channel Design

The serial input channel is essentially self-clocked logic controlled by a fundamental mode state machine. A state control design was selected because it was believed that a state machine could most easily interface between the asynchronous nature of the serial link, and the synchronous nature of the internally clocked logic and microprocessor interface. A block diagram schematic in Figure 18 represents the DI Serial Channel receiver circuit.

A number of basic features are incorporated in this chip. First, an even parity bit is appended to every data word for primary error checking. Second, a chip reset is provided to

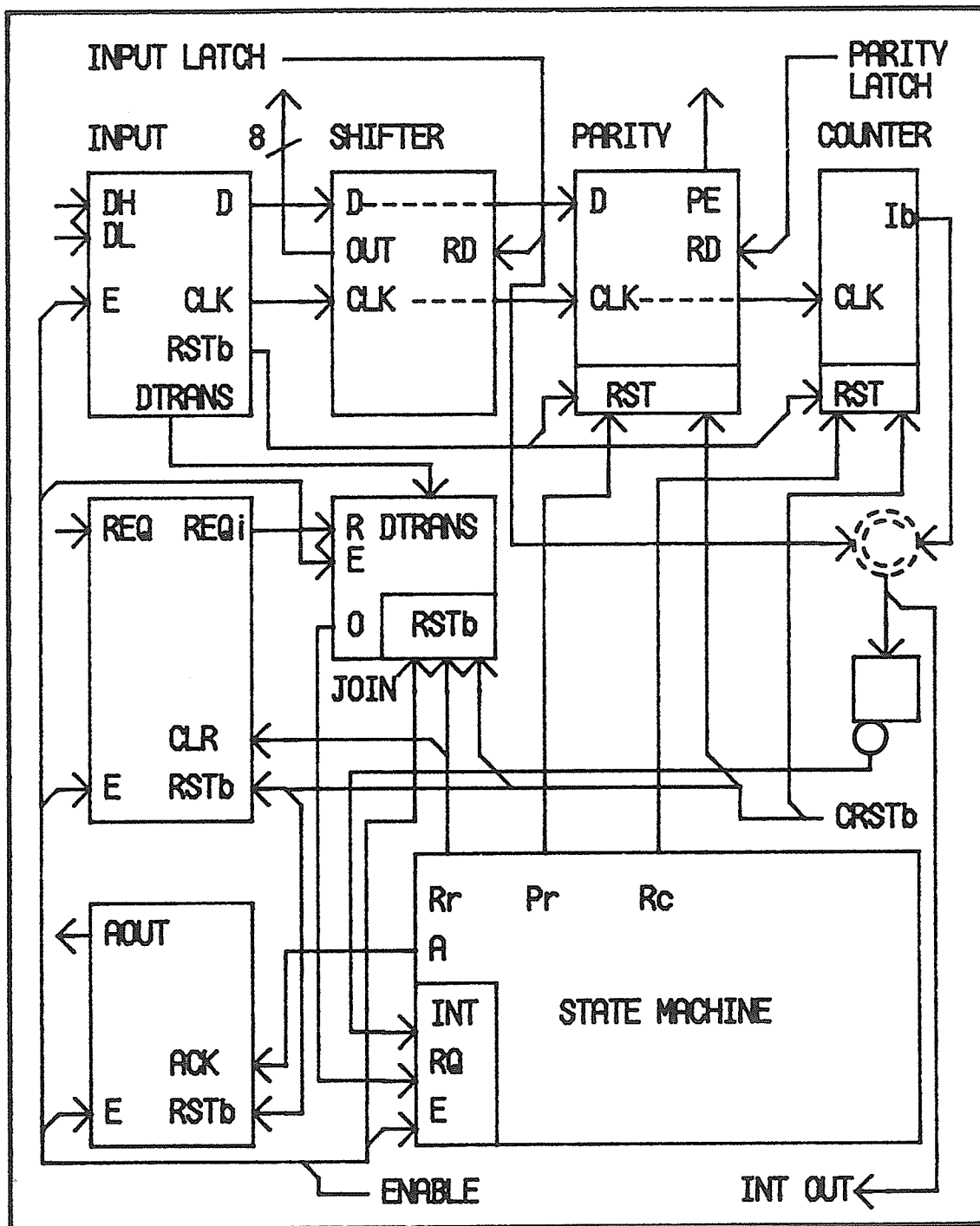


Figure 18: Input Channel

allow quick initialization of all channels on the IC to a known set-up state. Third, a channel enable register permits the selective reset and disabling of any input or output channel. Fourth, selection of an appropriate channel by either reading an input shift register, or writing to an output shift register, provides the means of responding to received data, or initiating data transmission. Finally, the chip drives an interrupt line to provide a microprocessor with interrupt driven communications; as well, both input and output channels are interrupt driven. The following sections shall detail the various functional blocks shown in Figure 18.

The input level generator receives DH and DL from off-chip, and produces a binary logical data value, as appropriate, as well as a synchronous clock edge signal. This edge provides all the needed clocking signals in the circuit. Additionally, an asynchronous reset is produced to provide appropriate internal conditions upon receiving a framing state. This block drives all the synchronous logic that directly receives data in the input state of the input channel. Finally, a completion signal (DTRANS) indicates when valid logical data is present at the inputs.

The shifter is a nine bit shift register and simply serves to convert the binary serial data into a parallel format. Note that the design assumes that the most significant bit of data arrives first. This circuit is a simple cascading of D type flip-flop circuits implemented in static CMOS.

The counter is four bits wide and serves to indicate when the input channel has provided nine consecutive input bits. At this point it signals the state machine that eight data bits and one parity bit are ready in the input channel. This allows the state machine to provide an interrupt to the system (indicating the presence of data) and to delay its ACK signal until the microprocessor has read the available data from the shift register.

The parity register simply provides an even parity check on the data. Included in this block is a tri-state driver to allow the latching of the parity bit onto the common IC data bus as part of a parity register.

The input request (InREQ) module is responsible for converting the double rail REQ signal into a form more acceptable to a fundamental mode state machine. It simplifies the state design considerably by having the double rail act as a single rail internally. Since transitions on an internal line are usually much faster than on external lines it was considered an acceptable compromise. The output of this stage is joined by a Muller C with the DTRANS signal from the input stage. The effect of this is to force the state machine to wait until all inputs are valid. This stage can be reset by a chip reset or by the enable signal.

The input acknowledge (InACK) module produces the double rail ACK output signal. This signal is produced by a toggle-type flip-flop initialized by the chip reset. This

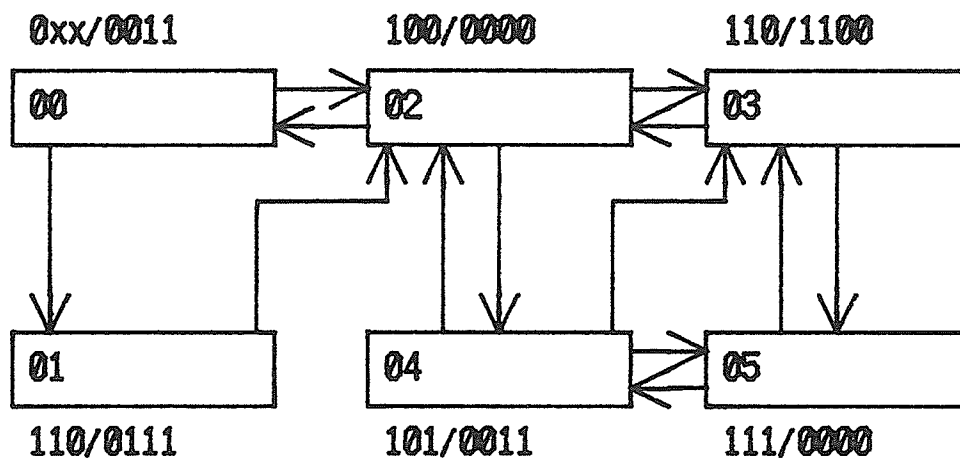
signal is driven by a state output of the controlling state machine. Thus, the state machine arbitrates the protocol by controlling the acknowledgements.

The state machine itself maintains the fundamental mode restriction; that is, only one input may change at one time. Careful design of the overall input channel and analysis of the various signals allowed the use of this restriction. Further, signals that could potentially be simultaneous are provided for in the state design. Design and analysis of the state machine has shown that this machine should be able to resist simultaneous input changes on all appropriate inputs from the various channel stages. Figure 19 details the state transition graph of the input state machine while Table II details signal functions. Design of the state transition table accounts for variation in ordering in INT and REQ; further, appropriate logic methodologies void critical races and hazards in the implemented circuit. One detail to notice is that acknowledge signals for both logical data and null adds considerable complexity to this machine. If the null state could be eliminated, considerable circuit simplification would be achieved for both the state machine and the input channel as a whole.

E RQ I / A Rr Pr Rc

E - ENABLE
RQ - REQUEST
I - INT PENDING
A - ACK OUT

Rr - REQ RESET
Pr - PARITY RESET
Rc - COUNTER RESET



NOTE: IT IS ASSUMED THAT ENABLE = 0 FORCES A RETURN TO STATE 00.

Figure 19: Input SM

TABLE II: INPUT CHANNEL SM SIGNAL FUNCTION

FUNDAMENTAL INPUTS

| | | |
|----|-------------------|-----------------------------|
| E | Enable Bit | 1 channel active |
| Rq | Input Request | 1 data pending |
| I | Interrupt Pending | 1 signals wait on processor |

FUNDAMENTAL CONTROL OUTPUTS

| | | |
|----|-----------------|--------------------------------|
| A | Acknowledge Out | 0->1 toggle acknowledge line |
| Rr | Request Reset | 0->1 reset request input to SM |
| Pr | Parity Reset | 0->1 reset parity register |
| Rc | Counter Reset | 0->1 reset control counter |

The remaining circuitry is primarily glue logic. All built-in test features and the overall microprocessor interface are not shown in this diagram. These aspects of the IC are presented in subsequent sections in this chapter.

3.3.2 Output Channel Design

The output serial channel is very similar to the input channel in design and scope. The major difference is the fact that this channel converts parallel data into a serial signal. Further, the synchronous clocking is provided by the output state machine in conjunction with ACK signals received from prior transmissions. Another major feature is that some action must initiate a transmission sequence. In this design it is appropriate that a write to the output channel shift register will initiate data transmission. The following paragraphs describe the output channel block diagram pictured in Figure

20.

The output level generator (Outdriver) receives a binary data signal, a data output strobe, and two control lines. The control lines serve to force either a null/frame signal, or a data signal, depending upon the value of the DCLK (data strobe) signal. The DCLK signal forces out data if its level is a 1, otherwise the OutDriver circuit will force out a null/frame signal as appropriate. Data input is via OH or OL which allow the circuit to output either the parity bit, or a data bit. The data clock signal is controlled by the output channel state machine.

The output shift register (S-REG) converts parallel data from the data bus into serial data for output on the serial channel. Data is written into the shift register by driving a negative edge on the input data-latch line. All data is shifted out with the most significant bit first.

The counter provides a count of the number of bits sent out on the serial channel. It serves to signal the OutDriver that the parity signal must be sent on the serial channel; thus, it allows arbitration between data output and parity output. Additionally, the state machine receives a signal indicating the status of the clock and the number of bits sent. The state machine uses this information, along with a delay to allow the circuit to settle, to provide arbitration and to control the internal clock generation. Please note that actions forced by the SM-generated clock must be allowed to settle before circuit

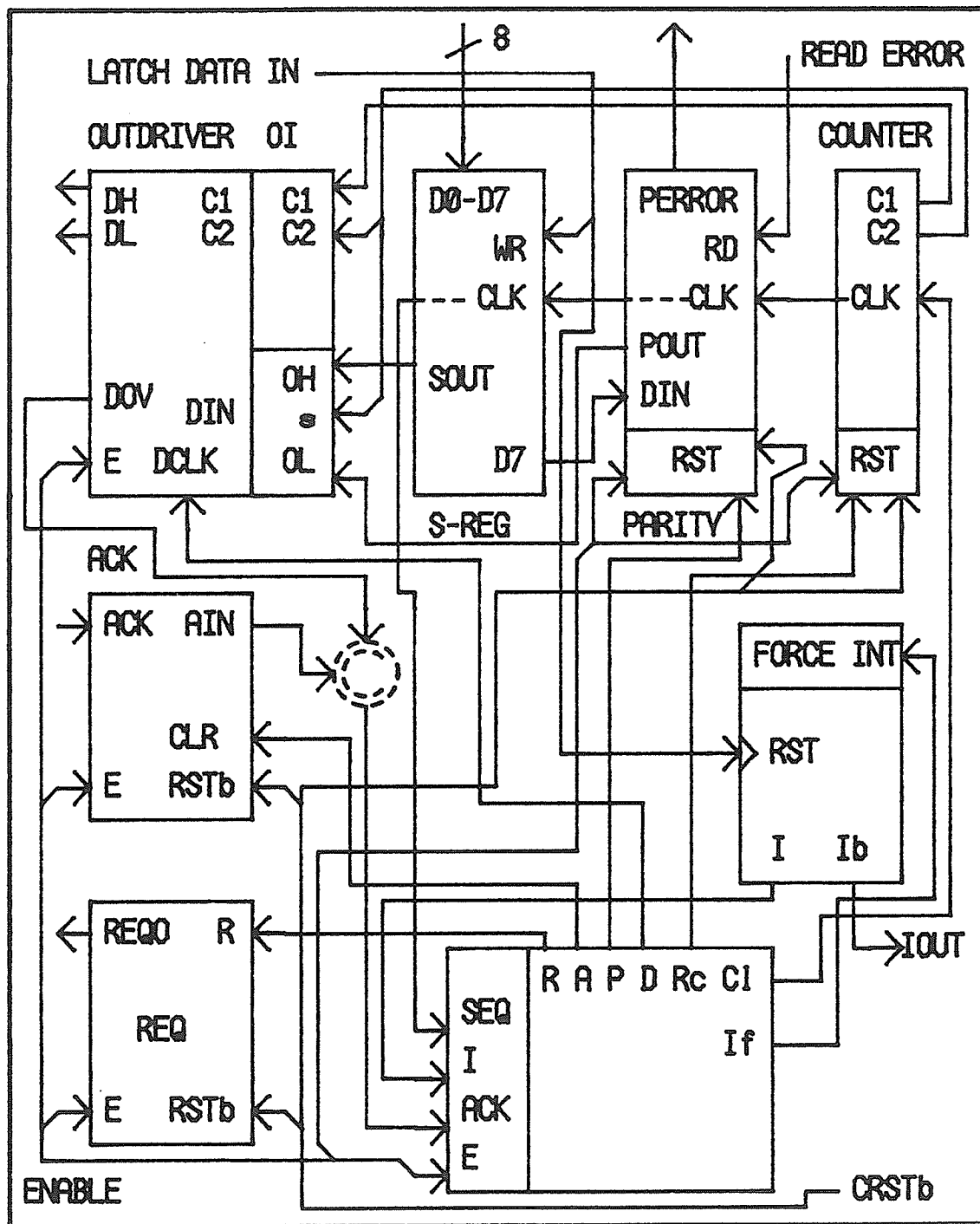


Figure 20: Output Channel

activity can be allowed to progress; the SEQ input is provided to assure this internal synchronization. In essence, it allows the design of a state machine using the fundamental mode restriction. Care was taken during state design to permit the possibility of simultaneous switching events where needed. Fundamental SM mode is modified only to allow for asynchronous events upon the Enable line since it was found to be a useful design choice. The state machine state transition graph for the implemented in Figure 21. Table III provides a summary of the SM signals.

TABLE III: OUTPUT CHANNEL SM SIGNAL FUNCTION

FUNDAMENTAL INPUTS

| | | |
|----|-------------------|------------------------------------|
| E | Enable Bit | 1 active |
| AQ | Acknowledge In | 1 acknowledge pending output |
| I | Interrupt Pending | 1 wait on external servicing |
| S | Sequence Signal | 0 wait on internal synchronization |

FUNDAMENTAL CONTROL OUTPUTS

| | | |
|----|-------------------|-------------------------------------|
| R | Request Out | 0->1 force request to toggle |
| A | Acknowledge Reset | 0->1 reset acknowledge SM input |
| Pr | Parity Reset | 0->1 reset parity register |
| D | Clock Data Out | 1 place data on DH DL, 0 place null |
| Rc | Reset Counter | 0->1 reset counter |
| Cl | Clock Shift Reg | 0->1 clock parallel to serial shift |
| If | Force Interrupt | 0->1 force a pending interrupt |

The ACK circuit converts the double rail ACK into a single rail circuit. This is joined with a data completion signal from the OutDriver circuit to simplify internal state machine

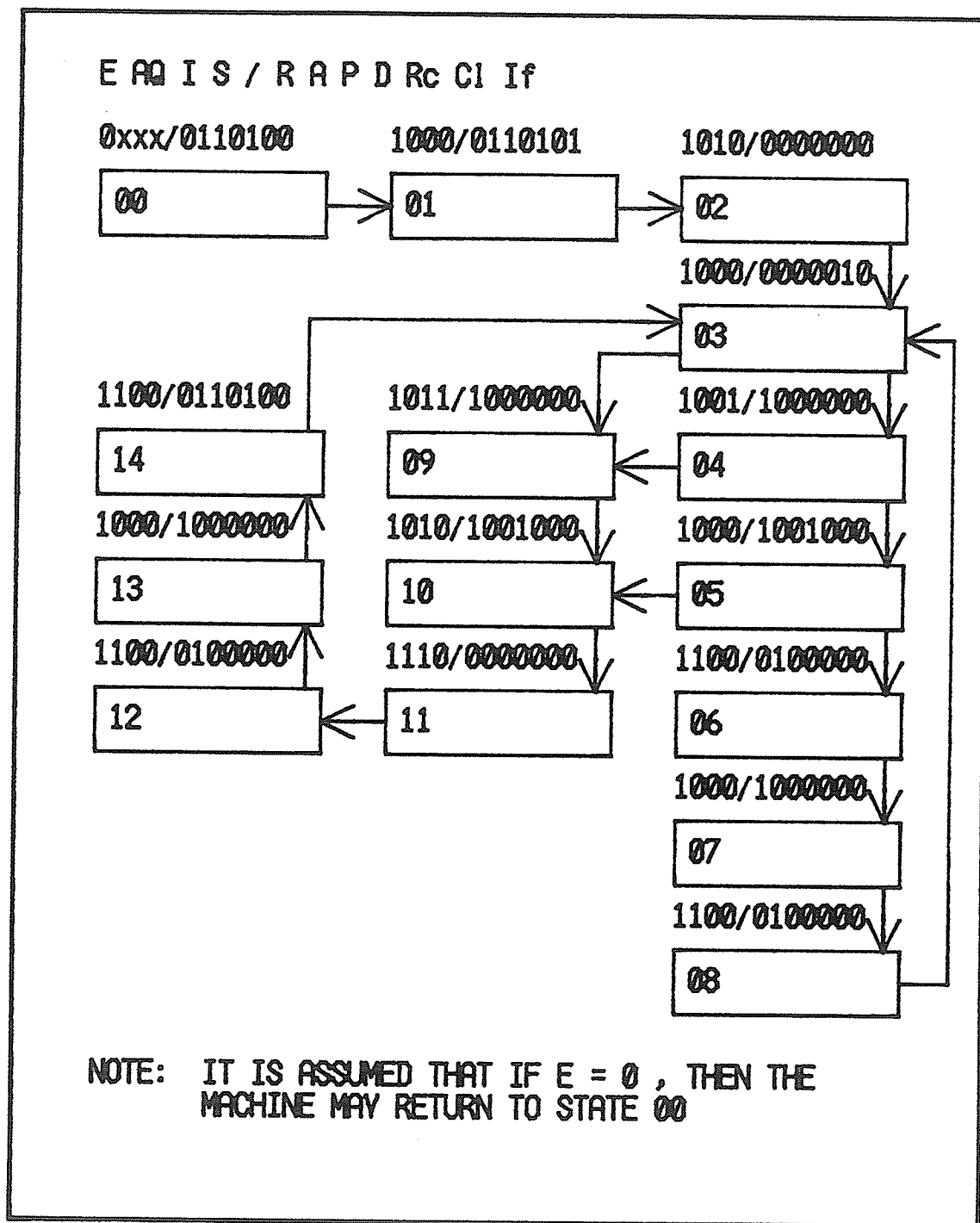


Figure 21: Output SM

design. The circuit is otherwise similar to the request circuit in the input channel. As well, the REQ circuit is virtually the same circuit as the ACK circuit in the input channel.

The remaining logic is primarily random logic within the circuit. Overall, the output channel was moderately more difficult to design than the receiver, and required a slightly larger area on the IC.

3.3.3 Muller C Design

The vast majority of the circuits in the IC use CMOS cells from the University of Manitoba VLSI cell library. Thus, most of the design uses D and T flip-flops, NANDs, NORs, and XORs. The one major exception is the Muller C, or Join, element. This logic element had to be produced for this design; further, it is not a standard static CMOS circuit. It is implemented using Pseudo-NMOS and is based upon an NMOS version presented in Mead and Conway [6]. The basic NMOS circuit is depicted in Figure 22. Since this circuit is a low level element and is critical to the reliable operation of the design, extensive simulation was essential. Therefore, I utilized the SPICE circuit simulator to gain an accurate view of its behavior under circuit loading of two to five gates. SPICE simulation indicated that the given circuit

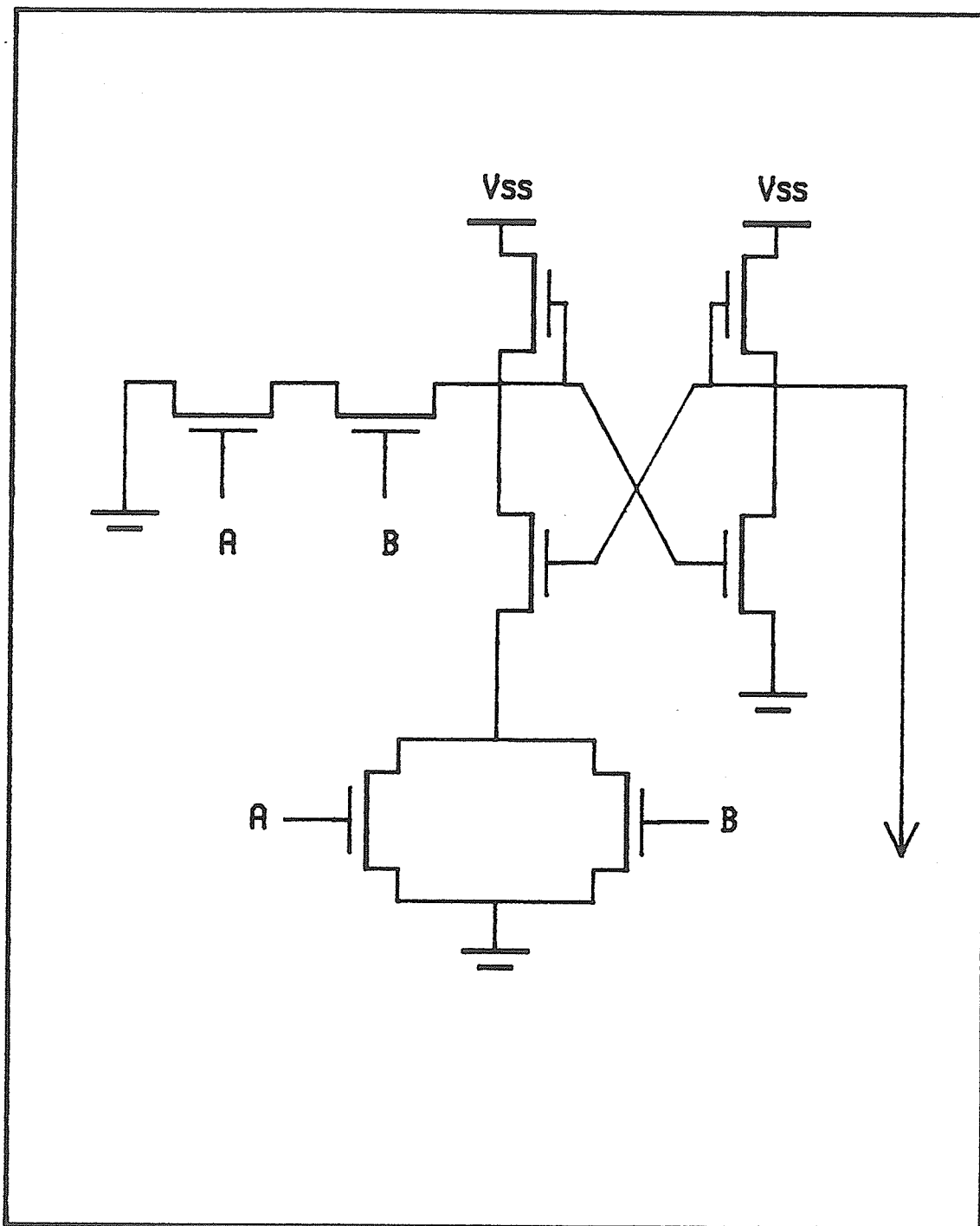


Figure 22: Muller C Circuit

implementation should produce an acceptable performance.

3.3.4 Microprocessor Interface

The DI serial IC provides a register-style interface to external processors. Access to internal registers, test points, and several other resources is via an eight bit data path. To read or write to this chip an external processor must assert the read/write line (R/W), two address lines, A_1 and A_0 , and an enable line. Such a configuration provides the user with a total of eight registers, four used for input and four used for output. Table IV summarizes the addressing for each register and its general purpose.

TABLE IV: INTERNAL REGISTER CONFIGURATION

| A_1 | A_0 | RW | FUNCTION |
|-------|-------|----|---|
| 0 | 0 | 1 | Channel 1 input register |
| 0 | 1 | 1 | Channel 2 input register |
| 1 | 0 | 1 | Internal test point for shift registers |
| 1 | 1 | 1 | Read parity register |
| 0 | 0 | 0 | Channel 1 output register |
| 0 | 1 | 0 | Channel 2 output register |
| 1 | 0 | 0 | Not currently used |
| 1 | 1 | 0 | Enable register |

The register set provided will be briefly described in the following text, while Figure 23 provides a complete description of individual register configurations. First, the two input channels, referred to as ic0 and ic1, are the eight bit serial devices on chip. They reside at the base of

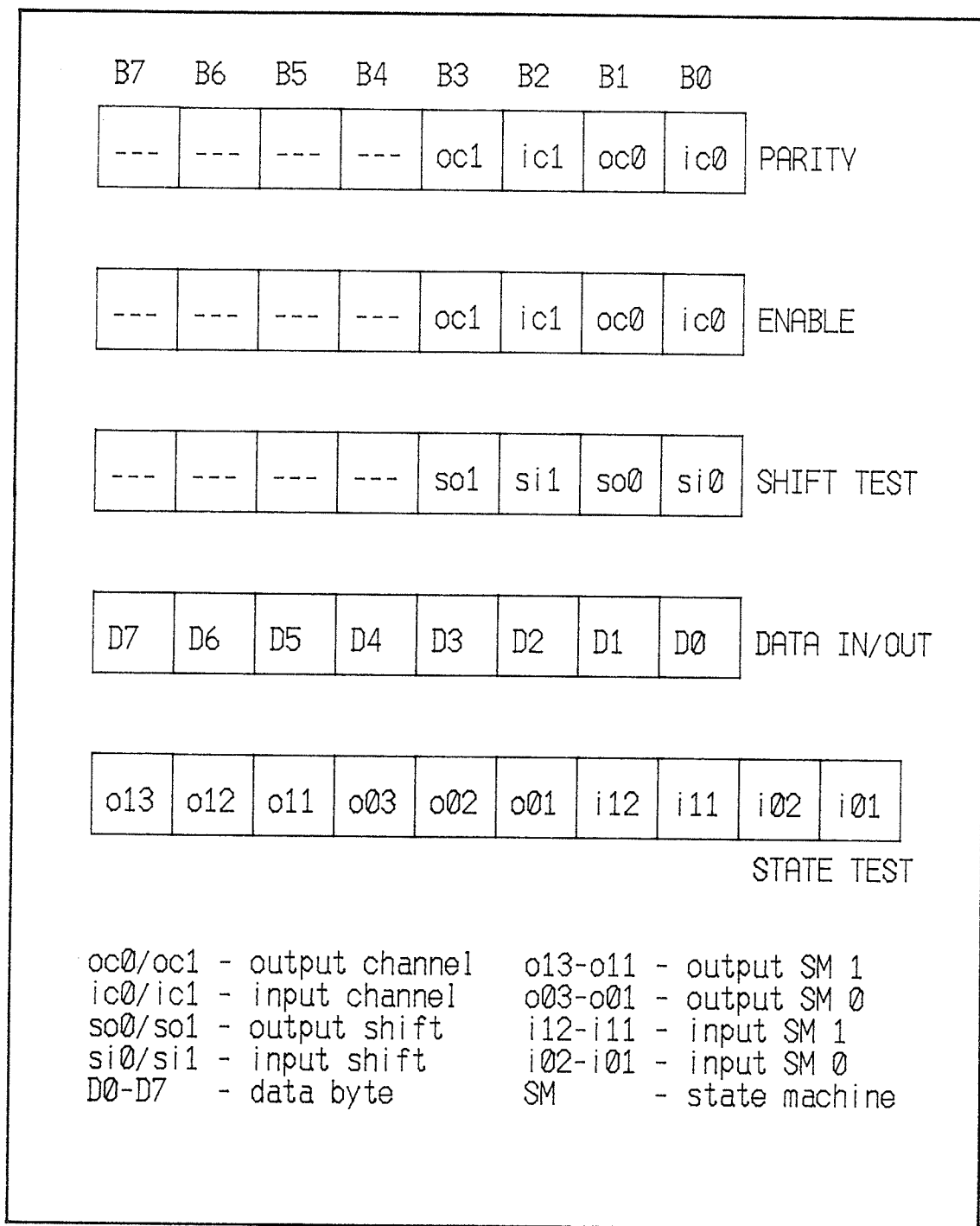


Figure 23: Register Ordering

the address range for read operations and are full eight bit parallel data registers. The two output channels, oc0 and oc1, are the eight bit parallel data outputs. These registers are at the base of the register set for write operations. All the above registers provide the device with the basic conversion from parallel to serial and vice-versa. The shift test register provides access to internal parts of the shift circuits and is intended solely for IC testing. The enable register allows us to individually enable or disable individual channels in either direction by loading a one or zero bit respectively; thus, we can disable the outgoing channel 1 while still being able to receive data on that channel. The parity register provides parity for both input and output. It provides both input and output parity bits in order to enhance the ease of testing for the output channel. The other registers listed in Figure 23 are used for testing purposes and are further discussed in the test section of this document.

3.3.5 Design Area and Layout

The asynchronous DI protocol IC implements two fully bidirectional serial channels on one chip 7518 by 7518 microm chip. The pad selected is a forty pin pad providing sufficient lines for the channels. Each channel, either sender or receiver, requires four pins to support the protocol and data

conventions. Thus, each and every fully bidirectional channel actually requires eight pins. The remaining IC pins are used for interrupts, parallel data, power, ground, and other microprocessor support functions. See Table V for a summary of the total pin count and usage. Unfortunately, even forty pins is quite limiting in this application since the two channels did not utilize the full chip area. In fact, only about 60% of the total chip area is utilized in the current design. Therefore, it would be desirable to sacrifice pad area for additional I/O pads by using a sixty pin package in future designs. Pad dictated arbitrated the choice of design layout and placement.

TABLE V: PIN USAGE

| Total Pins | Pin Name | Usage |
|------------|--------------------------------|--------------------------------------|
| 1 | SSI | Shift in state condition |
| 1 | SSO | Shift out state information |
| 1 | SCLK | Shift clock for test |
| 1 | LD/SH | Load or shift state |
| 1 | MUXCON | Control of state mux (normally 0) |
| 8 | D ₇ -D ₀ | Parallel data I/O |
| 2 | A ₁ -A ₀ | Address lines |
| 1 | Iic0 | Interrupt ic0 |
| 1 | Iic1 | Interrupt ic1 |
| 1 | Ioc0 | Interrupt oc0 |
| 1 | Ioc1 | Interrupt oc1 |
| 1 | RW | Read/Write |
| 1 | E | Chip enable |
| 1 | RST | Chip reset (active low) |
| 4 | ic0 | iOREQ, iOACK, iODH, iODL |
| 4 | ic1 | As per ic0 |
| 4 | oc0 | oOREQ, oOACK, oODH, oODL |
| 4 | oc1 | As per oc0 |
| 1 | VCC | Power |
| 1 | GND | Ground |

The implementation of the two bidirectional channels, the microprocessor interface logic, and the test circuitry in the ADIC IC utilizes approximately 60% of the total chip area available. In providing this area estimate it is only reasonable to mention that each channel is far from being packed as densely as possible. During the cell placement process I was acutely aware of the fact that pin availability limited this initial design. Many cells route input and output in configurations that did not provide efficient routing; hence, significant IC area is consumed by routing channels.

Major improvements can be made to the layout. Cell redesign to optimize inter-cell routing could greatly compress the individual channels in the current IC design. I estimate that tailoring cells more closely to final placement objectives could easily result in savings of ten to twenty percent over the current size of individual channels on this chip. Additionally, sacrificing some of the convenience of full CMOS for pseudo-NMOS could substantially reduce the size of many of the larger cells. If further reduction of area is needed, some attention could be given to providing a dynamic implementation of some of the blocks. Prime candidates for pseudo-NMOS, or dynamic logic, would be the shift registers, the microprocessor interface components, and any other logic where CMOS design is nonessential and the additional effort would yield significant compression in the circuit. Another major source of wasted area are the state machines themselves. Time constraints

forced me to implement the state machines in fundamental mode logic using NANDs, NORs, and NOTs. Implementation of the state machines using PLAs would substantially reduce the SM size which would, in turn, be a significant saving since these devices alone are about ten percent of the total channel area. Overall, I believe I could readily shrink individual channel area by twenty to thirty percent, which could result in a decrease in current IC area used to fifty percent or less. A single channel should drop from using about fifteen percent of chip area to about ten to twelve percent. Given the overall lack of packing of the individual cells such a reduction does not seem overly optimistic. Therefore, a more usable IC with four bidirectional channels should be possible using the sixty-eight pin package currently available from Northern Telecom. Table VI summarizes the chip layout statistics.

TABLE VI: DIE AREA USAGE

CURRENT DIE USAGE (60% of TOTAL IC AREA)

| | |
|-----|--------------------|
| 15% | State Machines |
| 20% | Shift Registers |
| 15% | Interface Logic |
| 10% | Glue Logic |
| 40% | Interblock Routing |

REDUCTIONS OF 30-40% COULD BE ACHIEVED BY:

- Cell redesign to improve routing
- PLA State Machine implementation
- Pseudo-NMOS/Dynamic logic blocks
- Improved placement

Of the channels themselves, it is important to note that only about twenty-five percent of the total area is exclusively

devoted to the DI protocols. This includes all the level generators, REQ/ACK circuitry, and the full state machine. In reality, the control of the DI protocol is fairly simple for the state machine, and it is the synchronous interface logic that greatly complicates the current design. Perhaps protocol simplification and greater adherence to DI properties in the state machine itself could greatly simplify this part of the logic design. Overall, the overhead added seems reasonable given the advantages a DI protocol can provide in simplicity and consistency of interconnection between serial channels. Please note that protocol overhead should be significantly less in designs that incorporate more DI features in their internal design rather than using a DI system simply added to an existing synchronous system. The ADIC IC design is isochronically clocked and uses highly synchronous design methodologies, while the state machine provides integration between modular DI asynchronism and the synchronous elements within a consistent framework. Chapter four will discuss alternatives to this design at some length.

3.4 Design for Testability

It was desired that there be some provision for testing provided within the IC implementing the DI asynchronous protocol. During the design process the options of built-in

self-test and external test were considered. Analysis showed that a high percentage of the signals are, or can be relatively easily made to be observable by using fairly simple methods. This is simply a consequence of the great independence of the various channels; testing is much like testing four simple chips rather than one large, highly interconnected chip. Therefore, it was decided to implement a combination of scan-type and adhoc methodologies for testing. The primary goal of testing was to initially determine defective chips rather than to isolate specific defects. The following sections will discuss measures taken to facilitate the testing procedure.

The primary goal of my efforts in testing was to make as many of the critical internal lines available to the tester as possible. One of the basic methods used was to make primary internal lines available to the tester. Therefore, the parity register includes the output parity bits to aid in testing the parity data path. Second, all shift register inputs and outputs are provided via a gated register for testing. Further, two other simple structures were added to facilitate testing, as described below.

First, it was considered convenient that state machine current state, and machine inputs, be directly observable. Thus, each set may be gated onto the data bus by driving the MUXCON input high (MUXCON has the dual purposes of controlling the SM multiplexor and the gating of state information onto the eight-bit data bus). Upon asserting this input line, the

address lines A_1 and A_0 will address the data according to Table VI. This method was deemed acceptable since it allowed the observation of a great number of the internal control lines for each device. Thus, testing could easily proceed by testing each channel individually by giving a simple check on the results produced by the primary state machine input circuits.

TABLE VII: STATE TEST ADDRESSES

| A_1 | A_0 | ST | CHANNEL STATE MACHINE ACCESSED |
|-------|-------|----|--------------------------------|
| 0 | 0 | 1 | Input channel 0 (ic0) |
| 0 | 1 | 1 | Input channel 1 (ic1) |
| 1 | 0 | 1 | Output channel 0 (oc0) |
| 1 | 1 | 1 | Output channel 1 (oc1) |
| x | x | 0 | Normal operation |

In a fundamental mode circuit the primary inputs and the state combine to produce the state machine output values. A simple scan-path type of test circuit is provided for further state control and testing as illustrated in Figure 24. A register and multiplexor arrangement allows us to either read the state machine state by using an appropriate combination of control values (drive MUXCON to 1, clock LD/SH by forcing in a 1, and then set LD/SH to 0) or by simply clocking the state test clock. This will effectively shift the current state machine states out of the SSO line for observation by the tester. As well, we can shift values into the state machine register by placing data on the SSI line and driving the shift clock correctly. In this manner we can literally force any

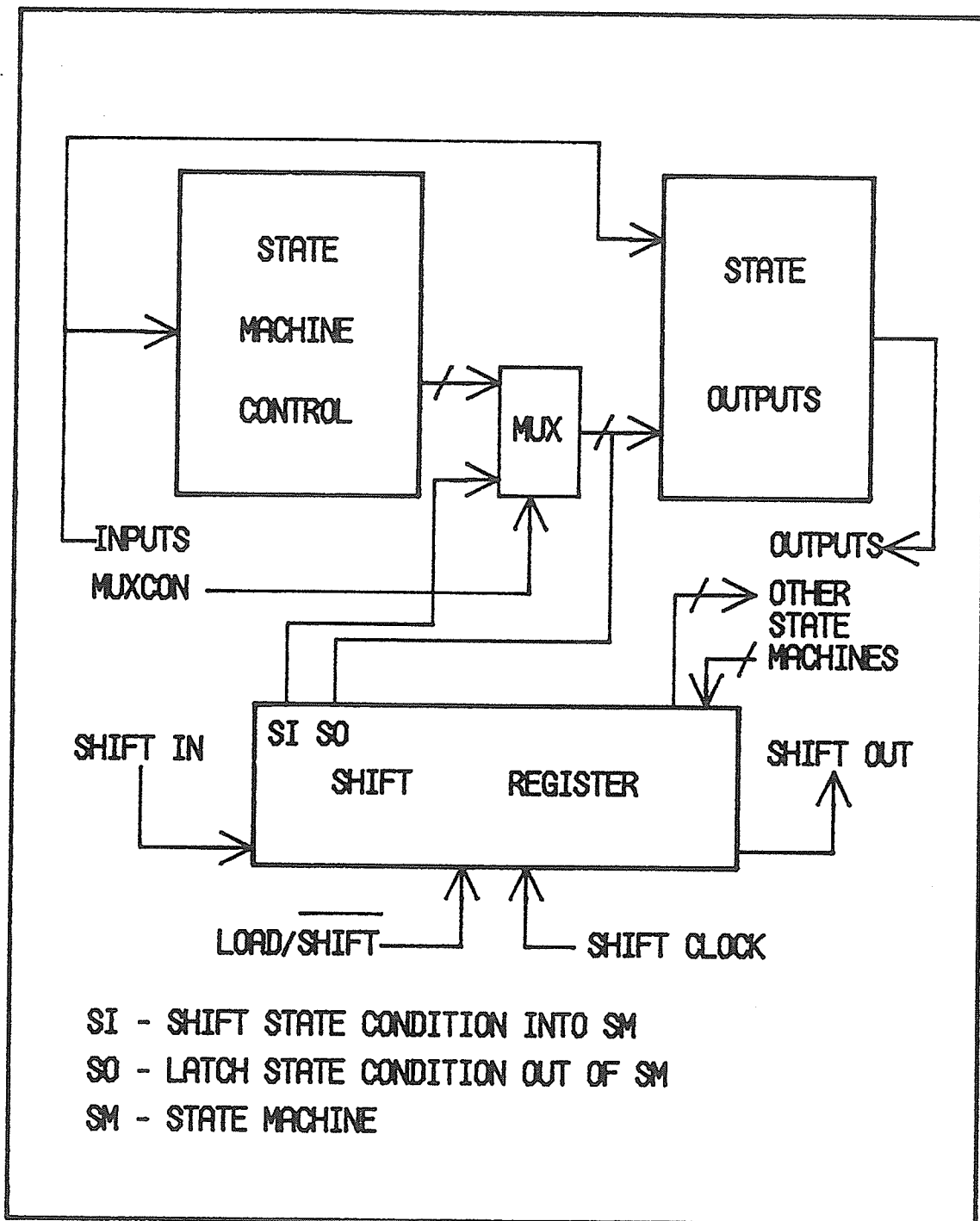


Figure 24: SM Observability

state machine output condition we desire by shifting into the state register and driving MUXCON low. Therefore, the user has the ability to observe the state outputs and can control the state outputs to test other parts of the machine. Figure 23 indicates the register format implemented on the current IC.

Extensive attention was paid to the state machine because almost all internal control lines, and most other vital lines terminate at the state machine. It was felt that greater observability and testing control could be gained most easily by adding more control at this level of the design.

3.5 Functional Simulation Results

3.5.1 Simulation Procedure

Prior to fabricating a chip of any complexity it is necessary to extensively simulate the performance of such an IC. The goal of simulation is to verify both the functional behavior and the overall dynamic performance. Extensive simulation provides greater confidence in the results of the design process prior to committing the design to semiconductor.

The simulators used in verifying the DIAC IC were the SPICE circuit simulator, the CSIM switch simulator, and the APLSIM simulator implemented at the University of Manitoba by Roland Schneider. SPICE is an extremely accurate simulator for

analog simulation. It was used to simulate low level circuits wherever timing or analog effects could be critical. SPICE could not be utilized to simulate the entire circuit because it imposes excessive demands upon computational resources. Thus, for larger blocks of logic, and for the overall functional and performance simulation, APLSIM and CSIM was extensively used. CSIM is a switch-level simulator which was used to provide a simple verification of logical connection and operation of logic blocks. APLSIM is quite reliable for overall simulation providing that the logic conforms to reasonable standards in logic levels and behavior. Therefore, SPICE was chosen to test analog effects at a low level, and APLSIM was chosen for overall simulation from the highest level to the lowest level in the design.

IC simulation started from small functional elements in the circuit hierarchy and moved to higher levels as blocks were linked together to form the total circuit. An incremental approach in design and simulation provided reasonable assurance that low levels of the circuit would perform as expected in the final design. Ultimately, simulation was needed at the IC level to verify overall operation. Since the IC can be divided into four highly independent channel elements, microprocessor interface logic, and test logic, the simulation never reached unmanageable circuit sizes. High level testing involved simulation of the microprocessor interface, and individual simulation of the I/O channels. The channels were individually

tested for functional and performance behavior. Final tests involved linking separate output and input channels together for a full test of functionality and performance.

The high level of testing provides a reasonable assurance of the correct behavior of the IC. Additionally, APLSIM allowed us to determine if timing and logical problems were likely to exist within the circuit. In fact, APLSIM allowed the early determination of serious timing errors in the input channel state machine thus strongly justifying its use.

3.5.2 Simulation Performance

Each channel was driven separately by external signals to initially determine the overall internal delay within the channel circuitry. Once the first simulation runs were completed, it was possible to fine-tune the simulation parameters to determine the overall channel performance. The test results shown here are for a system that assumes very small line delays and a relatively low channel load. Therefore, predictions of performance in a real system will have to factor-in overall line delay. As well, remember that all estimates include the fact that the data line is a four cycle REQ/ACK protocol as a result of sending both data and a data null.

Separate simulation of the input and output channels

provides the result listed in Table VII. Minor variations due to the layout are expected in the results. The delay rates shown in the table are peak performance Figures and assume instant response to outputs generated by the channel under test. Obviously this is not the case and channel to channel communication will involve circuit delay at both ends. As well, the data rates are burst rates and do not account for interrupt latency involved in servicing the DIAC IC. Overall, the performance indicated by the simulation is promising.

TABLE VIII: CHANNEL PERFORMANCE UNDER APLSIM

| | |
|------------------|-------|
| Input Channel 0 | 60 nS |
| Input Channel 1 | 65 nS |
| Output Channel 0 | 80 nS |
| Output Channel 1 | 75 nS |

When the simulation was run with an output and an input channel connected together, a fair measure of overall performance is presumed was obtained. Functional behavior was as expected. The overall response varied from channel to channel but the best rate was approximately 300 nS per bit in burst mode. Once again it is necessary to remind the reader that the line delay for this test was negligible.

Finally, I would like to note that performance will be affected by two factors. First, line delay and additional charge times under heavy loads will increase the time per bit. The burst bit rate can be estimated by using the simple relation of

$$t_B = 300 + 2 \times D \times P \times t_D \quad (\text{nS})$$

where t_B is the burst time per bit, t_D is the line delay per edge, and P is the number of rails for the ACK/REQ (in our case two), and D is the number line data transitions per data bit (in our case two for data/null). Second, overall bit rates are highly dependent upon interrupt latency in the target system. Note that the greater the line delay the lower the sustained bit rate will be. A measure of the overall time per bit transmitted can be found with the equation of the form

$$t_{BS} = (t_B \times 8 + t_{INT}) / 8 \quad (\text{nS})$$

where t_{BS} is the sustained bit time, and t_{INT} is the interrupt latency time. As can be seen, interrupt servicing rate can significantly reduce the transmission rate.

3.6 Test Results

3.6.1 Functional Testing

The first test of the DIAC IC was a functional test. This test was carried out on a test rig using prototype perf-board which has basic switches and LEDs. All outputs to the LEDs were buffered by drivers to avoid excessive loading of the IC outputs. The functional test consisted of exercising individual receivers, or transmitters, with appropriate driving signals and observing primary outputs. This type of testing is

acceptable for a number of reasons: (1) All internal logic is static CMOS and can therefore operate at any arbitrarily low frequency. (2) Functional blocks on the IC are relatively small and are easily isolated for testing. (3) Basic functional tests can fairly easily verify correct operation. (4) On-chip testing paths improved the testability of the channels and greatly simplified the overall functional test. Of course, such a simple testing method cannot easily locate the cause of a fault but it can verify overall logical operation in this particular IC. This is not true, in general, for all ICs nor is it implied that this was an exhaustive test.

Initially, all of the ICs failed the functional test. Observation of the primary outputs of the ICs gave an indication of why the devices were performing so poorly--the logic levels were inadequate. Although some ICs were producing relatively high and low voltage values, as appropriate, noise margins and levels were totally inadequate. Replacement of the buffers by proper low power CMOS parts allowed testing to proceed for some devices. With no loading, or low loading with one low power CMOS load, proper voltage levels were finally obtained with some devices. Under such conditions, three of my five devices utterly failed the functional test. Of these devices, one had a non-functional input channel 1 and 2, another appeared totally non-functional, and the last had three of four channels totally non-functional. Of the remaining two, one barely maintained adequate levels on channel 1, and the

second was acceptable only for low load conditions.

Why did the devices originally fail? Examination of the simulation results indicated that the simulation was carried out with excessively low loads. True TTL loading was not properly applied to the outputs of the devices; the result is that the IC output driver stages are not adequate for greater than a low CMOS fanout. Although this is not entirely disastrous (one device did work), it does require the IC to have buffers placed between all outputs and their corresponding loads. As well, it would tend to limit yields. This certainly underlines the essential need to subject a device to realistic conditions at the IC boundaries during simulation.

3.6.2 Dynamic Testing

Dynamic performance testing was carried out on a Motorola MC68HC11 prototype module. This module has a microcontroller, a ROM, and interface circuitry on a small board. As well, a cable connector allows the interfacing of external boards to the prototype module bus directly. To test the device, wire was directly wrapped to cable connector pins so that the board could directly interface to the DIAC IC on a perfboard. A number of LEDs formed a simple diagnostic display on the perfboard. In order to measure the performance, one input and one output channel were simply tied together on a single IC.

Thus, I tested the interconnection performance for a small local wire length. See Figure 25 for an illustration of the test system. A simple interrupt service program was written to provide servicing of the input and output channels used. Diagnostic information about program and system operation were displayed on a simple four-segment intelligent display.

Performance measurement was accomplished in a very direct fashion. An Arion 100 MHz logic state analyzer was connected to REQ, ACK, DH, DL, RW, and INT which has an effective capture rate of 50 MHz for six inputs. By triggering on the interrupt line, a full data cycle could be observed for the IC. The main reason for using the logic state analyzer, however, was so that the peak performance could be directly measured.

Direct measurement provided a bit rate of approximately 1150 nS per bit (870 Kbps). This was a disappointing result but it was not surprising given the poor driving characteristics discovered in the functional testing of the ICs. Direct observation of the line with an oscilloscope was carried out by externally triggering the scope from the interrupt line. This provided a stable enough image to clearly see that excessive charge/discharge times appeared to be the problem. The IC spent most of its time charging or discharging the external loads, while internal operations appeared to require relatively little time. This is the expected interpretation since the time spent at a stable level was quite short which is in keeping with the simulation predictions of

SUSTAINED BI-DIRECTIONAL TRANSMISSION TEST

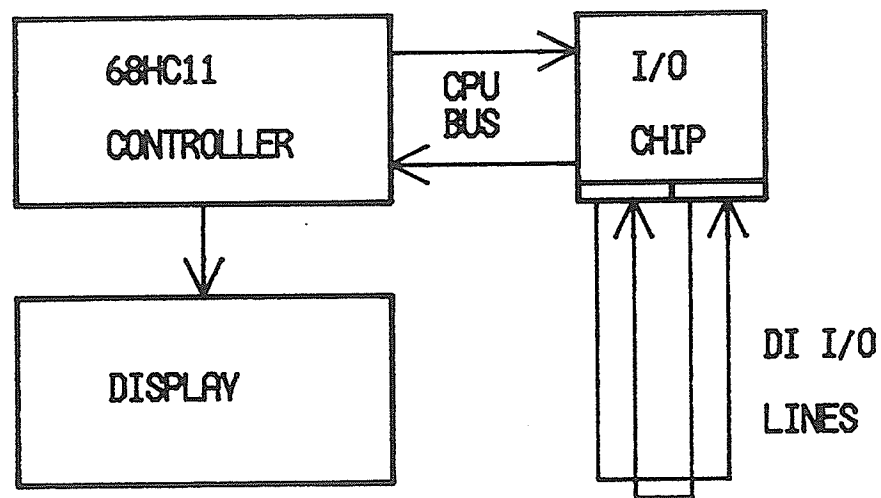


Figure 25: Test System

150 nS for a full cycle. It is somewhat pleasing that the IC functioned at all given the severe drive problems it experienced.

Finally, a delay line was produced using D flip-flops and a One Shot IC. The REQ line was delayed by 5000 nS to see if the device still operated. It did continue to function correctly with the expected 10000 nS additional delay per bit (remember, two edges occur on REQ in each cycle) for a total transfer time of 11190 nS. Note that the discrepancy between these measurements and those of the logic state analyzer is due to the fact that the delay is not exactly 5000 nS.

3.7 Evaluation

Unfortunately, the current IC proves to be inadequate due to its weak output driver circuits. Therefore, any attempt to use this IC in practical circuits would require new output drivers to be added to the IC.

Nonetheless, the IC does operate in a delay insensitive manner. In fact, it is probably the delay insensitivity of the internal logic that allowed it to operate with such inadequate output drivers. Thus, it does demonstrate that DI techniques have some merit. The error in simulation does not really alter the fact that the DI IC adjusted to its environment automatically when large line delay was inserted at the chip

boundary.

Further study could include redesign of this IC. A new IC should incorporate proper drivers, a two cycle REQ/ACK and data protocol, and internal mixed mode or fully DI circuitry. As well, attention could be paid to reducing channel area so that a more practical number of channels could be placed in the 64 pin Northern Telecom package.

CHAPTER 4: MACROCELL DESIGN

4.1 Why Another Design Method?

I believe that it would be desirable to utilize another design method for the implementation of DI systems for a number of reasons. The current design was an ad-hoc solution of a specific DI protocol. Unfortunately, this type of design is quite difficult, being expensive both in time and effort. Further, the specific protocol implemented in the ADIC IC is not as efficient as it could be; therefore, it is desirable to utilize a more efficient protocol specification. This is particularly important if DI methodologies are to be used at a lower level within logic circuitry than was carried out in this specific design.

Therefore, a more consistent, regular method of design is desirable for implementing DI systems and DI protocols between major functional blocks of logic. A more suitable design methodology should provide shorter design cycles and may well be more amenable to design automation. In particular, it should be remembered that using DI modules at a low level greatly simplifies interconnections since time critical paths

between blocks are of little concern.

4.2 A Logic Design Methodology

C. E. Molnar and others [3] present a general method for the implementation of DI specifications. In that paper, both a generalized formal circuit specification and a method for circuit implementation are presented. The method consists of three distinct steps: (1) For a given circuit, an interface state graph (ISG) must be produced. (2) An enhanced ISG (EISG) is then derived from the primary circuit specification ISG. (3) State equations are derived from the EISG. (4) These equations are then implemented in a DI macrocell. Thus, if a complete specification for a circuit can be produced, the tools are available to produce state equations to implement the functional behavior.

The core of this process is the derivation of an Interface State Graph to implement the procedure desired. The ISG is related to Petri Nets [5]. Like a Petri Net, an ISG requires a monotonic transition on a line which is signified by a graph variable. Variable transition is referred to as firing the variable. This variable activity is represented by arcs in the graph. For an ISG to make a change of state, a variable must carry out a firing event. A circuit implementing the ISG must, therefore, provide for monotonic transitions of each variable

represented in the ISG. Figure 26 is the proper ISG of the Muller C element mentioned previously as specified by Molnar and others [3][4][7]. In an ISG, a state node (represented by a circle) with two or more outgoing arrows represents a choice between one of two or more behavioral options. A state node with two or more incoming arrows represents a state reachable by many behavioral paths. If a state node has one input and one output, a clear sequence of events is explicitly indicated--first the input must fire and then one specific output must fire. This is the situation if we have a full ordering of the input signals. It is the goal of the ISG to specify all valid partial and full orderings of inputs and outputs in the DI circuit. Any number of inputs or outputs may be specified, but a circuit must output its data prior to accepting new inputs. Thus, a C element ISG indicates that A or B may fire in any order, but both must fire before the output, C, undertaking an output transition. Thus, A and B are partially ordered signals since they may be arbitrarily reordered prior to a valid change in the output. It is useful to note that an ISG may assume an initial state that is achieved either asynchronously or via state inputs in practical circuits.

An EISG is produced by giving each state node a binary code such that all inputs and outputs are represented by a unique binary symbol, and a start state in the ISG is assigned the state value 0000...00. The designer traces through the network by inverting the current variable state each time a related

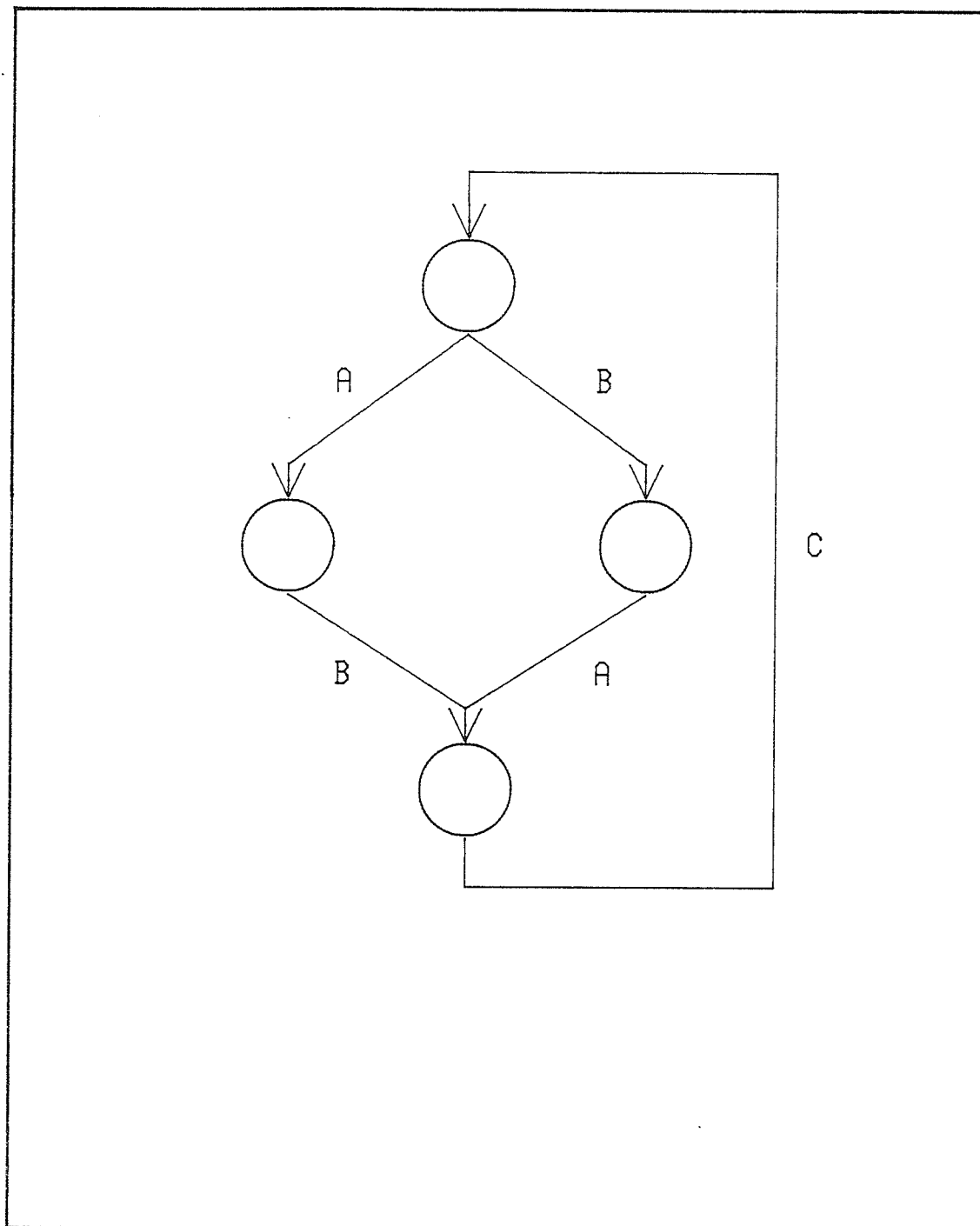


Figure 26: ISG of C Element

trace arc is encountered in the ISG. The designer must continue to traverse the original network, extending it as needed, until he traverses all of the traces and produces a result identical to the starting state bit pattern. It is normal procedure to begin with a bit pattern of all zeros, but this is not essential--it is merely a convenient consistency. Figure 27 illustrates the EISG of the Muller C element's ISG portrayed in Figure 26. Notice that the function is essentially duplicated in this case; in many other situations, the designer may find redundancy in the EISG paths.

Now it is possible to produce a state implementation of this function. Simply produce a boolean function for the state node values using a Karnaugh map (or use any other Boolean simplification method). In this specific case, we have an eight-valued, three-variable Karnaugh map. Wherever the function produces a '1' output (the C element in the preceding Figure 27), a 1 is placed in the Karnaugh map. From the map illustrated in Figure 28 (A) a logic function producing the desired result is readily implemented as shown in Figure 28 (B).

Clearly this is a bit simplistic as an approach for DI systems in general. It is well known that a designer could produce hazards, races, and so on, within the circuits in the process of converting directly to logic in this way. It is necessary to provide circuitry to prevent metastable behavior and common circuit component variation from causing severe

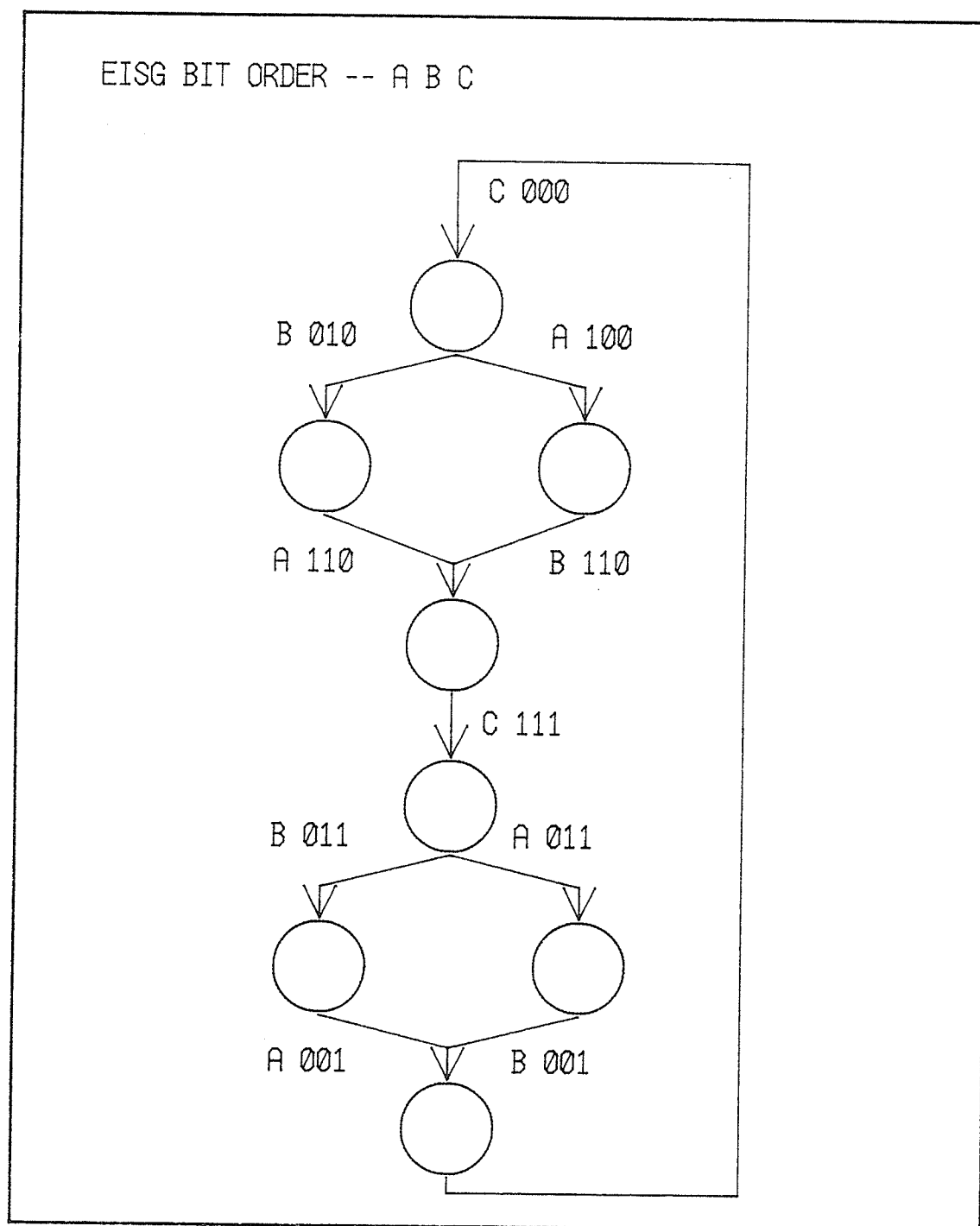
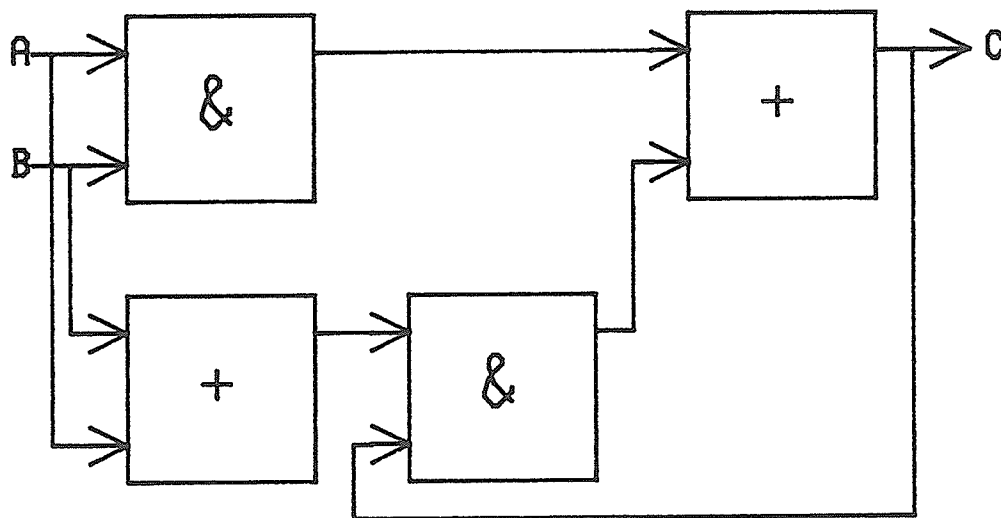


Figure 27: EISG of C Element

| | | ab | | | |
|---|---|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| c | 0 | 0 | 0 | 1 | 0 |
| | 1 | 0 | 1 | 1 | 1 |

$$F = AB + CA + CB = AB + C(A + B)$$

(A) KARNAUGH MAP OF THE C ELEMENT



(B) LOGIC IMPLEMENTATION OF C ELEMENT

Figure 28: C Element Logic

operational difficulties. The next section describes a hardware methodology that satisfies the DI requirements at the circuit level.

4.3 A Hardware Implementation of DI State Equations

A major design consideration in state machines, and many state-dependent implementations of DI systems, is that a number of basic circuit problems may occur. First, function hazards may be produced by delays in the feedback logic of the state machine implementations. Second, 'simultaneous' changes in state inputs can lead to logical and functional critical races. Unfortunately, this usually leads designers to specify fundamental mode restrictions or to use other approaches that limit circuit behavior. Removal of the fundamental mode restriction greatly complicates the design of most state machines unless a clocking methodology is employed. Such approaches are generally unacceptable in DI design since partial reordering of inputs and other characteristics must be easily accommodated. Further, state table solutions to hazards, races and similar problems are excessively difficult to implement in realistic circuits. Rosenberger et. al. [8] have suggested an alternative that will satisfy the DI requirements and provide reasonable design freedom for the state approach outlined in preceding sections.

A fundamental method for controlling state machines, is to place registers at all state inputs and outputs, and rely upon clocking to latch input variables and output results. The use of registers provides a more reliable, more easily implemented state machine for the designer. The only time-sensitive aspects of the design are the requirements that an appropriate delay occur between the clocking of input events and the production of valid outputs, and that the registered inputs have an acceptable setup time prior to the clocking events themselves. Unfortunately, changing a signal at the moment of clocking can produce undesirable metastable or unstable behavior in such a circuit [8]. Rosenberger has proposed a self-clocked state design called the Q-Module that can provide a DI implementation method that provides a robust environment suitable for the demands of metastability tolerant, non-fundamental mode state machines for DI systems. The goal of the Q-module is to simplify the task of the DI module designer to merely specifying the ISG, EISG, and equations, while freeing him from the difficulty of making standard state machines robust enough to behave reliably in a DI environment.

A Q-Module (see Figure 29) consists of Q-Flops, a state machine, a Muller C element, and a clock generator. A Q-Flop is a special flip-flop organized to form an input and output register. A state machine in a Q-Module accepts all inputs and outputs, and places output and state information into storage Q-Flops. A Rendezvous circuit (a multiple Muller C) and a

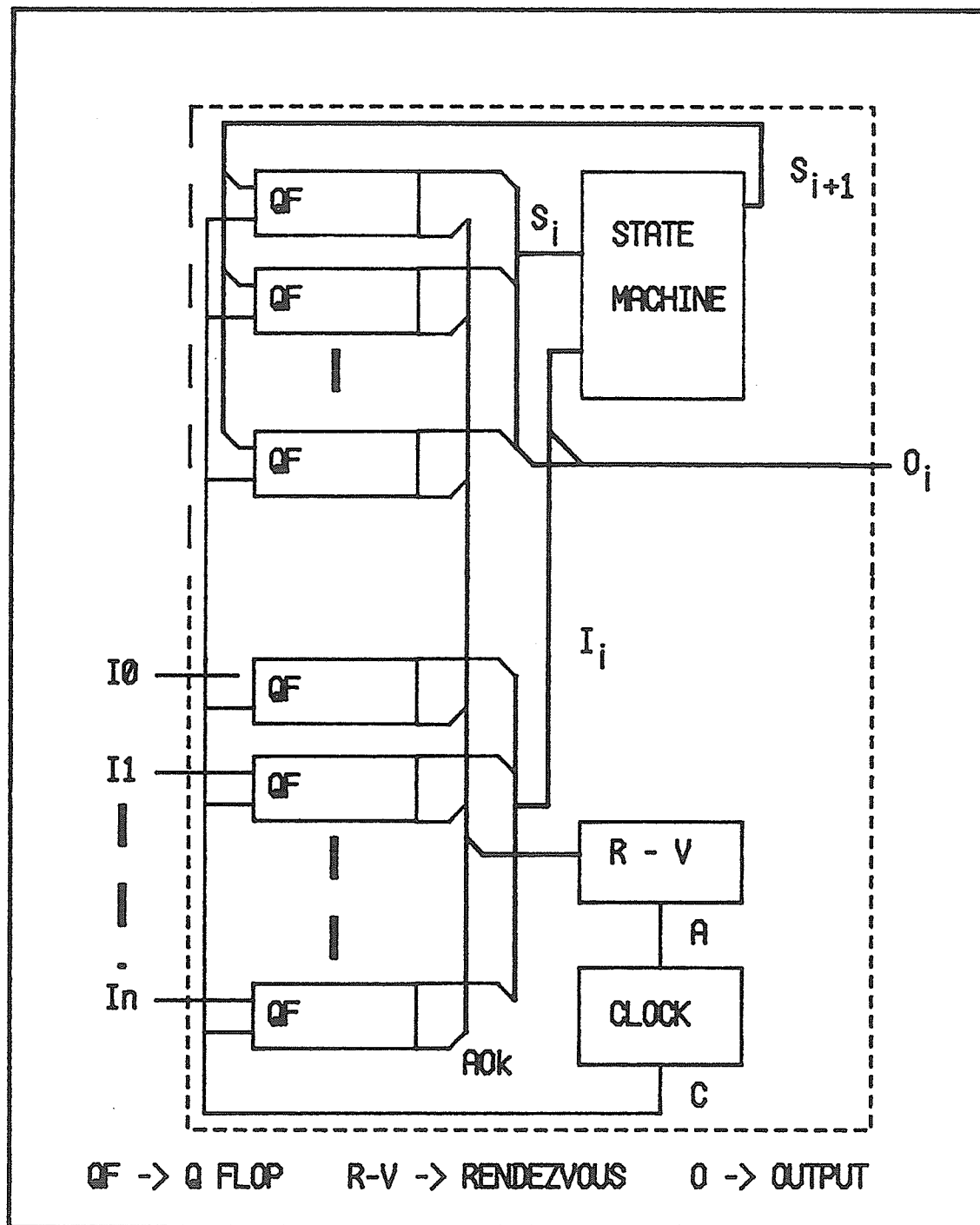


Figure 29: Q Module

clock generator provide the clocking for the Q-Flops. The critical features of a Q-Module are the unique behavior of Q-Flops and the circuit Q-Flop clocks.

The Q-Flop itself has a number of interesting features. The most important is the fact that metastable behavior of the input, such as an input changing at the instant of clocking, only delays an output signal. Hazards or other circuit instabilities cannot occur as has been verified by Rosenberger et. al. [8]. Secondly, the Q-Flop provides a completion signal, that is, it drives an acknowledge line when the output of the Q-Flop has been successfully updated. This acknowledge signal is a double rail signal and is consistent with DI signal specifications. The Q-Flop is implemented as two parts, a Q-Flop resolver and an output generator, as depicted in Figure 30. Rosenberger provides a number of circuits that produce the Resolver/Output pair by using thirty-seven transistors in NMOS or Pseudo-NMOS. It is the non-metastable nature of the Q-Flop that guarantees the proper activity of the DI state equation by assuring the presentation of valid inputs to the machine, while the Q-Flop acknowledge signal provides valid machine clocking.

The Q-Flop clocking method is also quite unique. Whenever the Q-Flop clock makes a low transition, the Q-Flop resolver samples the input signal. The resolver will not signal that data is valid until the input reaches a valid stable state; it provides such signaling by using a pseudo-tri-state line internally. When the clock for the Q-Flop carries out a high

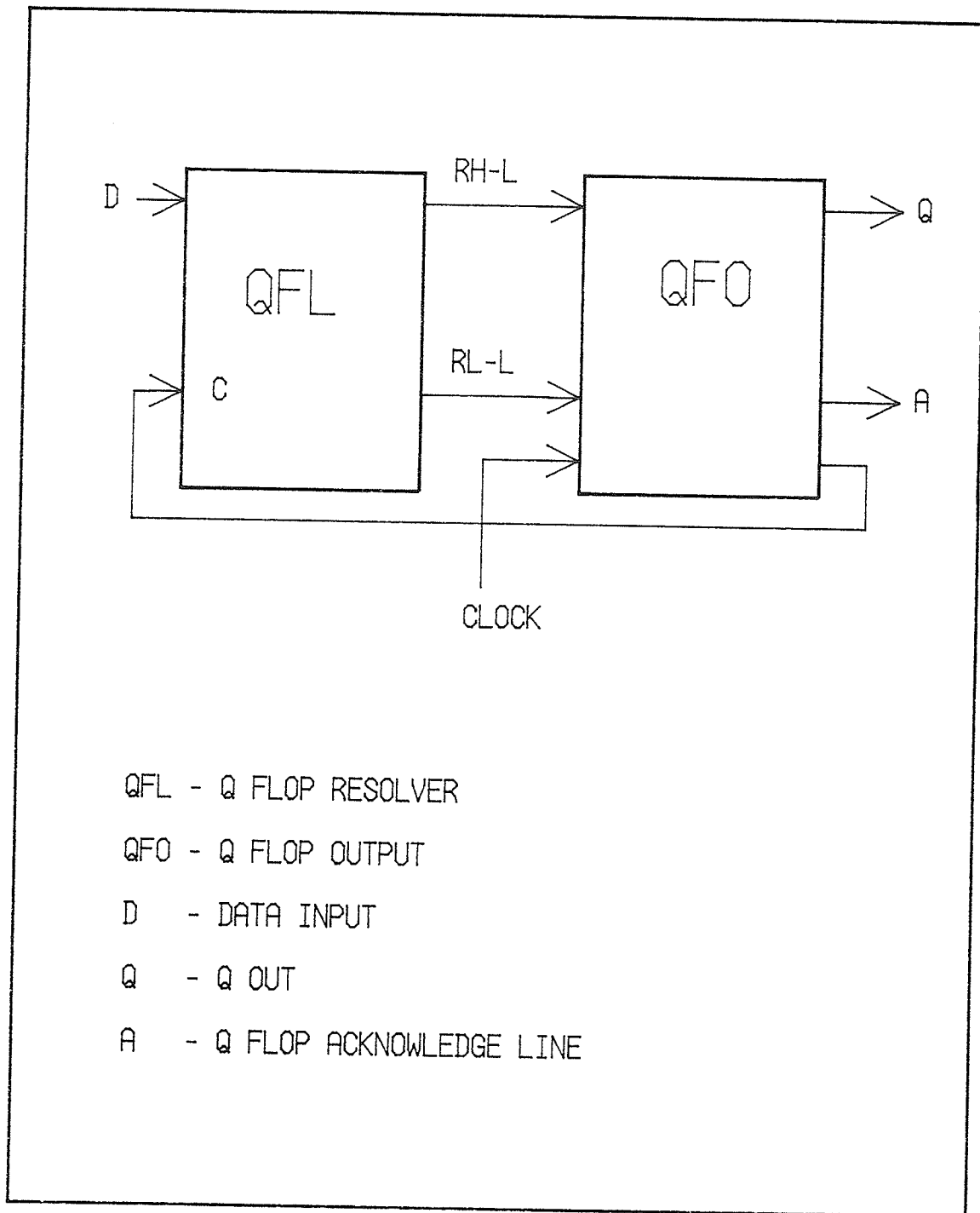


Figure 30: Q Flop

transition, the Q-Flop output stage will place the resolver value on the Q-Flop output lines. Thus, input to output in the Q-Flop requires one full clock cycle where one major assumption is made; the Q-Flop signals the validity of output data, therefore the clock must wait for the Q-Flop acknowledge before clocking continues.

Within a Q-Module, a specific clock signal is continuously generated. This Q-Clock has the edge organization pictured in Figure 31. It consists of a store input edge, CS, followed by an edge that will store the state machine outputs, CO. The signals that drive this clock are the individual acknowledges, A_k , from the Q-Flops themselves. The clock issues a store event, CS, so that new inputs may be stored into the Q-Flop resolvers. At this time, the old outputs will still be in effect; thus, only inputs can vary at the CS edge. This will clock data into the Q-Flop resolver circuits, but all outputs will still be stable. After a delay dependent upon the assertion of all Q-Flop A_k s, the clock will produce a clock output edge, CO, and store all of the valid outputs. At this point, the outputs will be placed on circuit output lines and into the environment. In practical terms, a state output is functionally delayed by one internal clock cycle since an input takes one clock edge to get to the state machine, and then one clock edge to reach the output Q-Flops. In this design the circuit must satisfy only one critical delay; there must be sufficient a logic delay between AO and CS

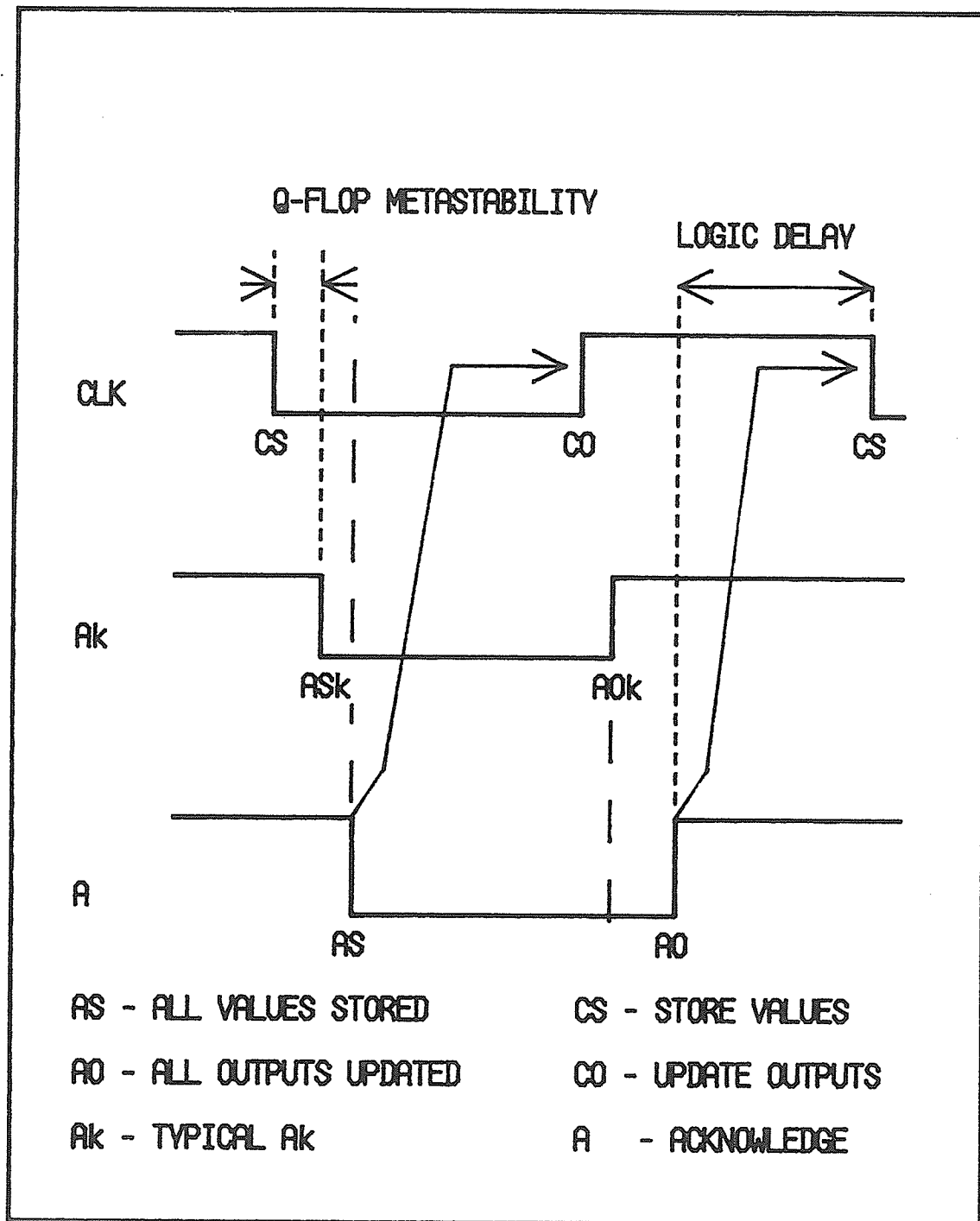


Figure 31: Q Module Waveform

to guarantee that the input Q-Flop results have been processed by the state machine prior to clocking results into the output Q-Flops. Other than this one critical delay, no other delays must be produced to ensure proper circuit operation. Finally, it may be convenient to ensure that the Q-Module starts at a known state. Such a condition may be easily achieved with minimal circuit overhead; simply logical AND the Q-Flop inputs with a global reset line to ensure preloading the Q-Flop with a reset state. Note that this method corresponds very well with the assumption that the EISG base state has state variables all equal to zero. Refer to Figure 32 for an illustration of a single Q-Flop reset circuit.

The next section will discuss a specific implementation method for a DI protocol. In particular, an extension will be offered for integrating sequential blocks into the current protocol methodologies.

4.3.1 Construction of an EISG Protocol Receiver

In this section we shall describe the design of a circuit that accepts a standard DI protocol and provides appropriate REQ/ACK signalling. The design methodology outlined in the previous section shall be used to produce.

Observation of the protocol in Figure 33 shows that a full double rail protocol cycle consists of either DL and REQ

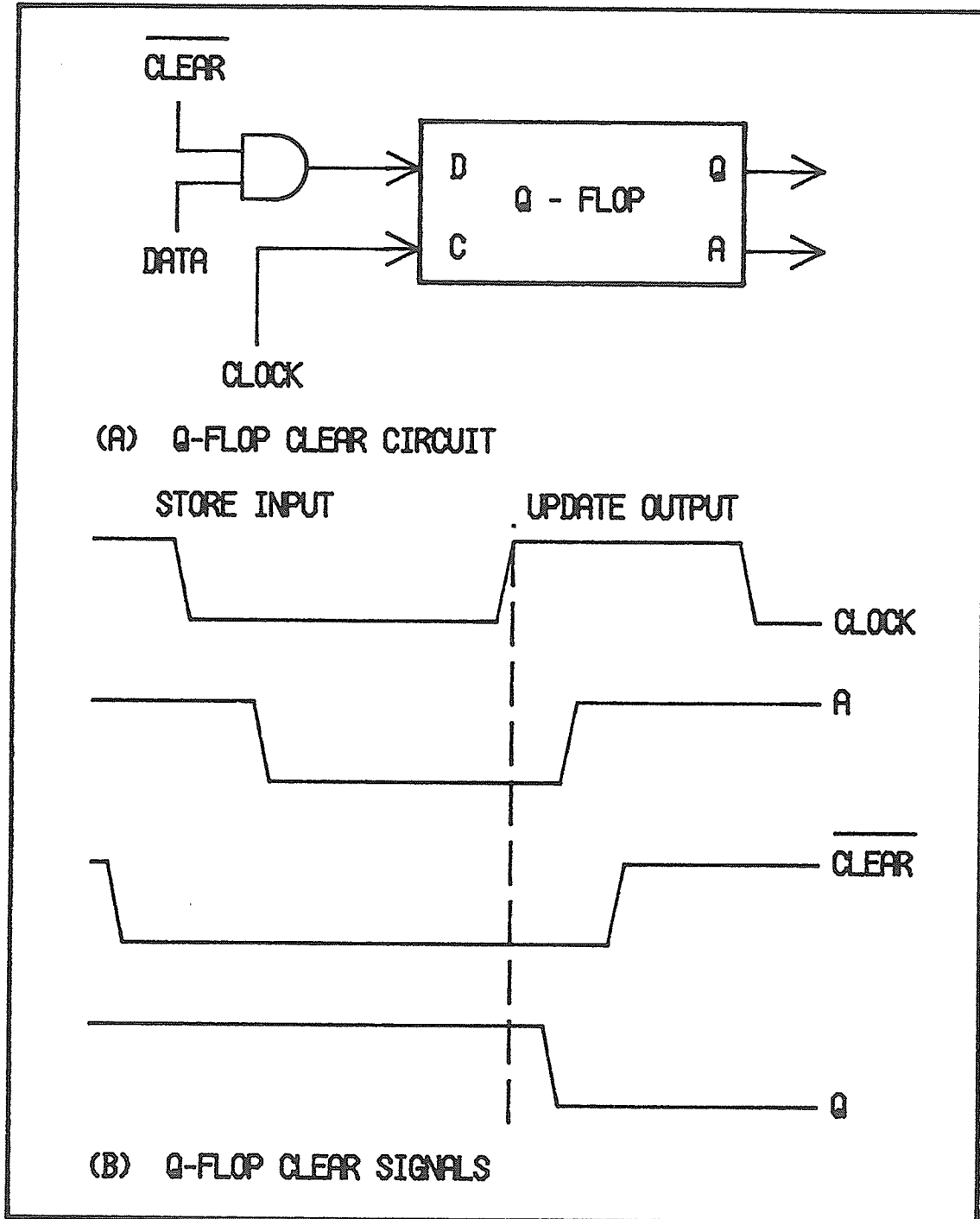
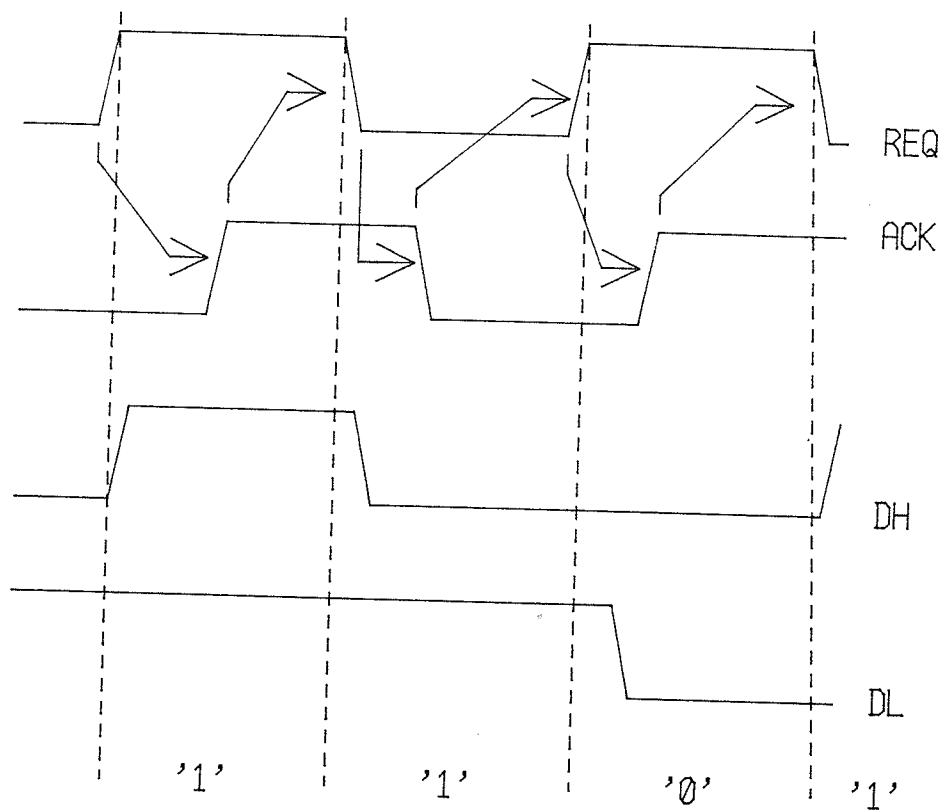


Figure 32: Q Flop Reset



NOTE: DATA MAY LEAD/LAG AN EDGE -- THE PROTOCOL
MUST ACCOMODATE THE SKEW OF THESE SIGNALS.

Figure 33: Two Rail Protocol

followed by ACK, or DH and REQ followed by ACK. As indicated, the data line and request are only partially ordered in this protocol, as would be expected in any robust DI system. This is a fairly simple cycle behavior and an interface state graph can be readily be derived that satisfies the requirements of a full DI interface. The graph in Figure 34 fully specifies this system. If we consult the protocol waveforms, it can be seen that both the waveforms and the DI reordering are satisfied by the graph pictured.

Extension of the graph by the method given in Section 4.2 results in the EISG depicted in Figure 35. This graph included extensive simplifications where duplicate paths resulted from the initial ISG graph extension. Note that duplication is amenable to simplification and the pictured graph is so simplified. This protocol EISG represents the full state representation for producing a DI protocol receiver having the above behavior. It is interesting to note that simplification after ISG extension reduced the number of nodes originally produced by about six, which leads one to wonder whether redundancy is a common situation with more elaborate ISG representations. In any case, the acknowledge signal is the output of this EISG and it may be easily encoded by the Karnaugh map technique mentioned earlier. The Karnaugh map of the EISG is portrayed in Figure 36 and it has been verified for proper operation by using the PALASM logic equation simulation tool produced by Monolithic Memories Corp. The equation in

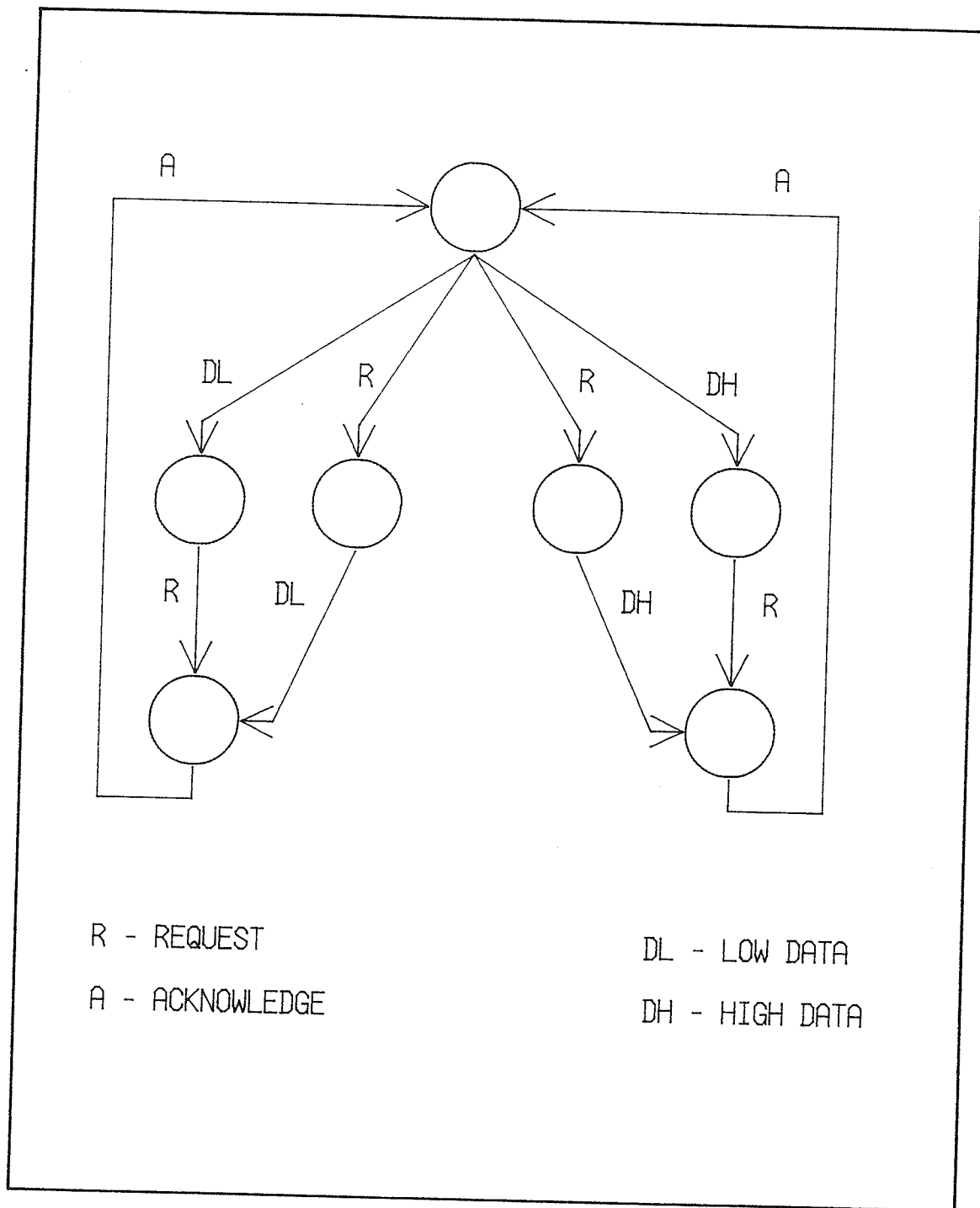


Figure 34: Protocol ISG

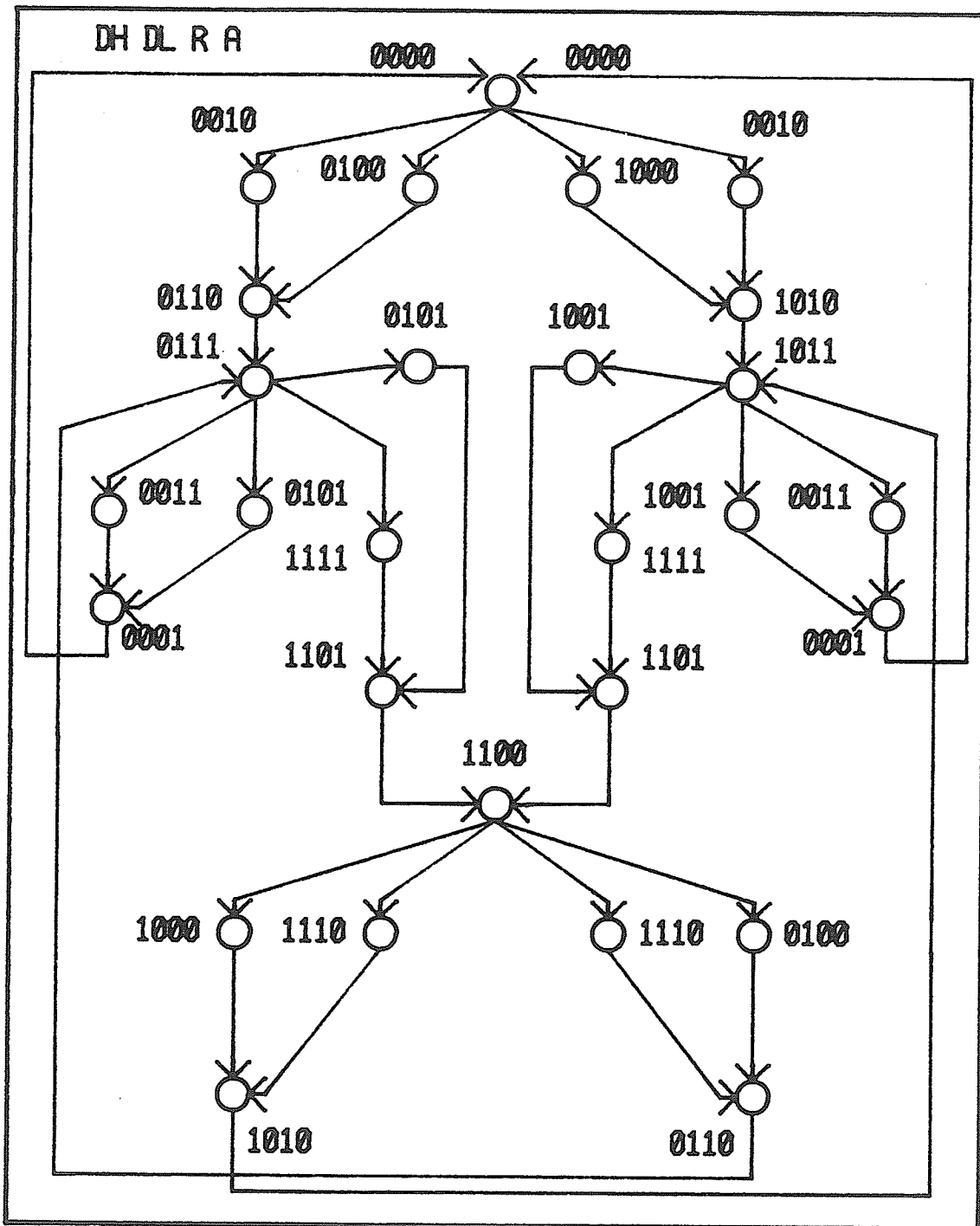


Figure 35: EISG of Receiver

figure 36 does implement the functional specification detailed by the ISG in Figure 34. Therefore, the formal method produced by Rosenberger, Molnar, and others [2][3] provided a simple and elegant solution for implementing this DI protocol. Implementation by Q-Modules would be straightforward.

For a given task, it is fairly simple to extend the functionality for this DI Module. A designer could insert the ISG of a data manipulation function into the protocol ISG just prior to the firing of the ACK signal. This would provide an operation prior to acknowledging the module's readiness for more data. The design would be completed by re-expressing the EISG and rederiving the logic equations for this circuit. It is probable that a complete Q-Flop module in which the function is inserted into the ISG at the outset would be more efficient than multiple modules. This is because much of the circuit overhead, such as clock generators, would not need to be repeated.

Alternately, this DI Module could be used as the interface to more extensive DI logic that performs computational or other tasks. For example, in a design like the asynchronous communication IC, we could cascade the DI input protocol into DI implementations of shift registers and logic interface control blocks. As well, there is really no requirement that future stages follow the protocol which this device implements; it is simply a functional block utilizing an enhanced protocol

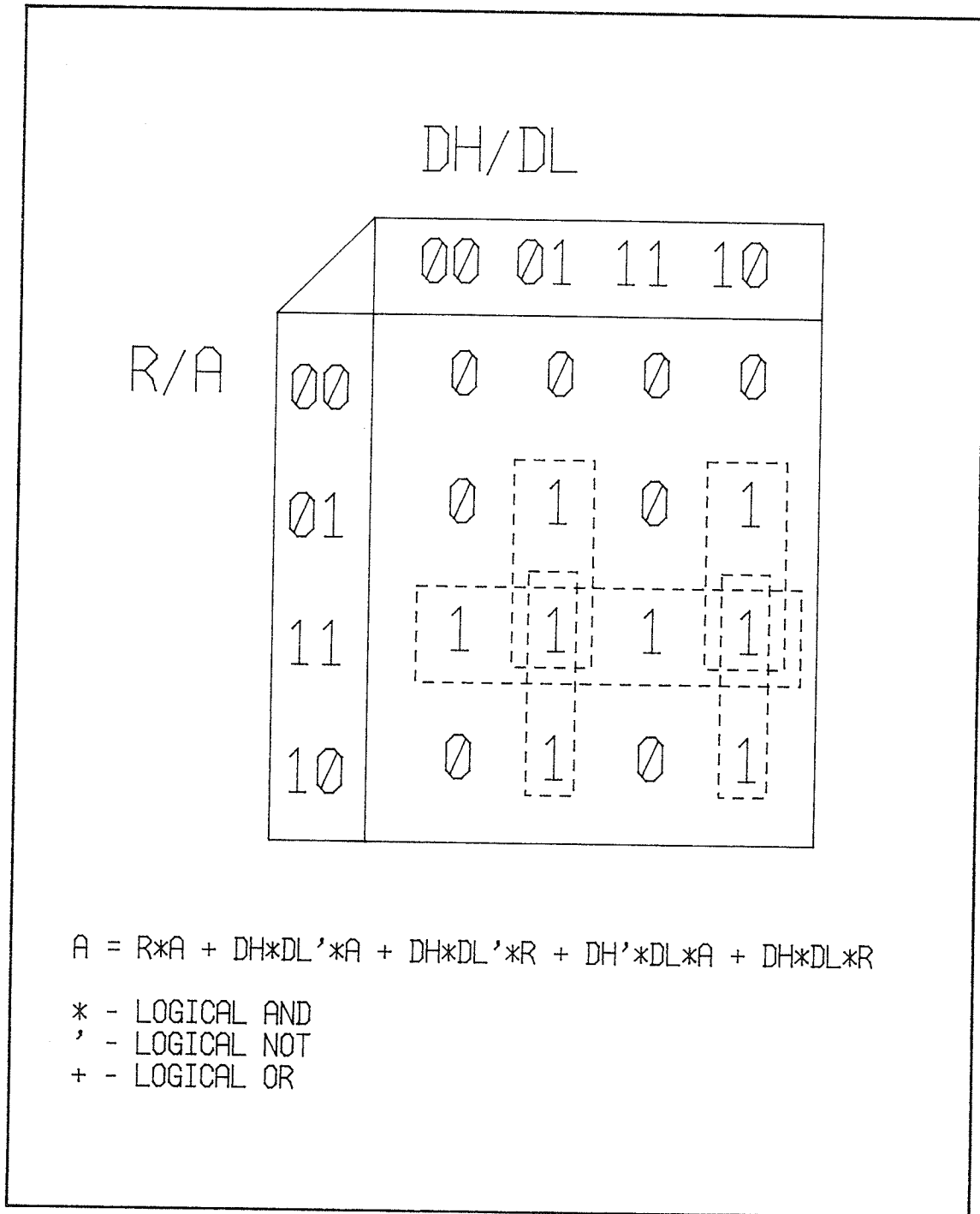


Figure 36: EISG Logic Map

Fover the one used for the ADIC IC produced in this project. All that is required is that each stage use the basic rules established in earlier sections of this document for parallel and serial operation of DI stages.

Principally, this DI receiver is intended for interfacing of synchronous isochronic logic within a DI environment. The reasoning is quite simple--clocked design in a limited isochronic zone is well understood and quite efficient. As well, cell libraries would initially be lacking DI components; hence, creating cell libraries would slow initial designs. An approach of gradual cell library creation would probably be preferable in most facilities. As well, the design of reliable synchronous systems over short IC distances is quite possible. It is the design of synchronization networks for long IC line distances that is increasingly difficult and prone to failure. Clock delays over long traces, clock distribution problems, and other details of signal skew are essentially solved by DI realizations. Therefore, the DI protocol specified above could be used for major long distance/time signal transmission while local design could still be isochronic.

Ultimately, the goal is to eliminate the clock altogether. However, it is likely that clocking will be used in the majority of logic for some time yet. Therefore, a mixture of DI and synchronous logic may have great merit for cost and ease of design. The type of design that mixes synchronous and DI logic shall be referred to as mixed mode design.

4.3.2 Mixed Mode Design

I will call the combination of fully DI elements and sequential logic in a DI module a "mixed mode" implementation methodology. The essential idea, as shown in Figure 37, is to use a DI receiver circuit to provide clocking to local sequential logic; thus, a DI protocol would be used between logical blocks while the designer has the option of implementing the needed functions using sequential logic. The clocking pulse, or DI ACK, is produced only after all appropriate inputs and REQ signals have been received. Parallel data can be handled by a simple parallel concatenation of serial stages using Muller C elements, or by proper redesign of the serial protocol for parallel data. If parallel data is desired, we need only have one set of DH/DL lines for each logical data line, as well as one REQ line. The logical extension of the ISG is relatively straightforward and merely involves more parallel paths. For example, an ALU could be the block following the DI interface and it could be implemented in sequential logic. All that is needed is an interface that provides the ALU with an appropriate clock signal. This design is free from global assumptions about clocking and synchronization beyond the DI standards while allowing the designer to locally optimize isochronic circuits. All that matters to a system designer is that the block itself be globally DI, and therefore it must globally support a DI

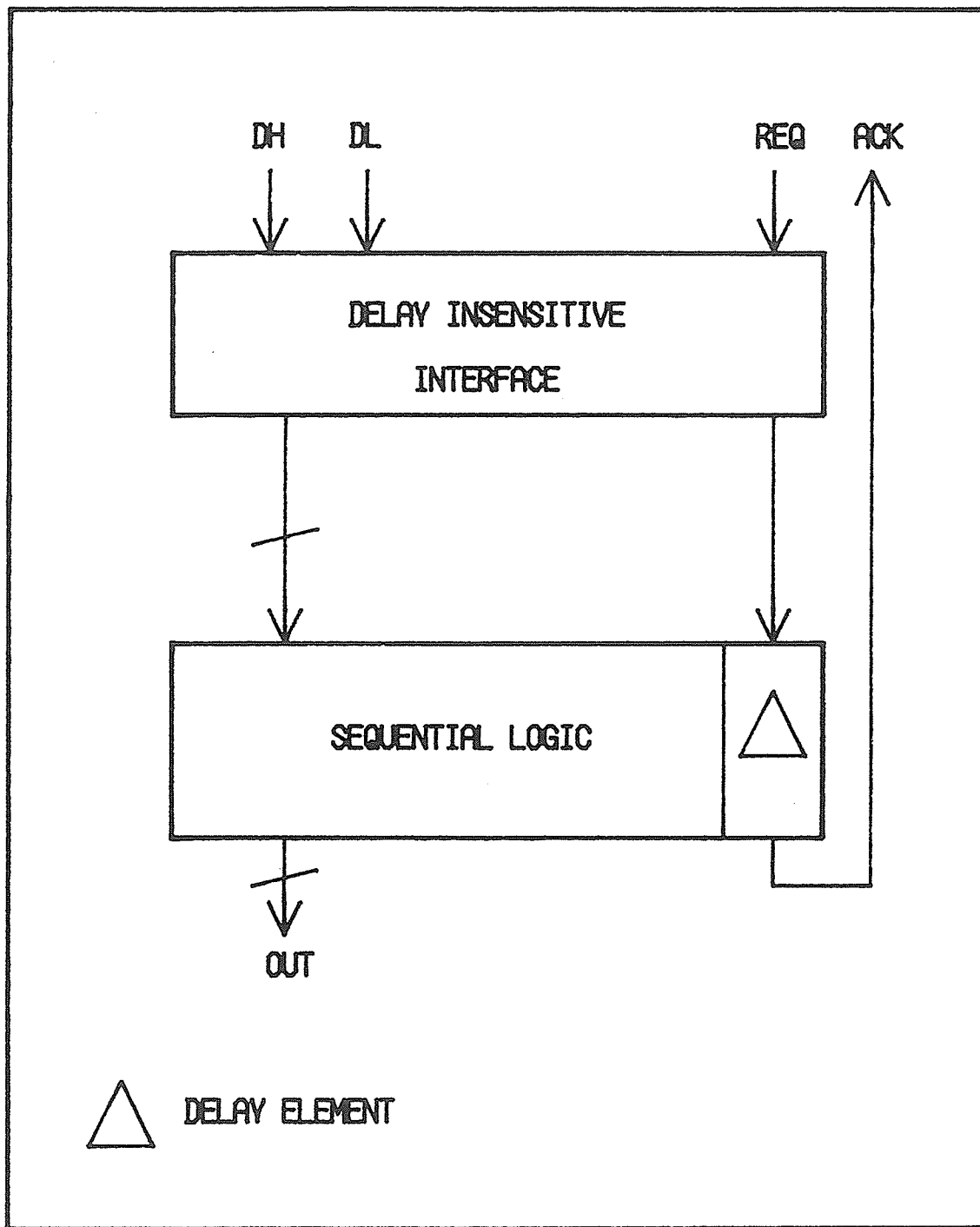


Figure 37: DI Receiver

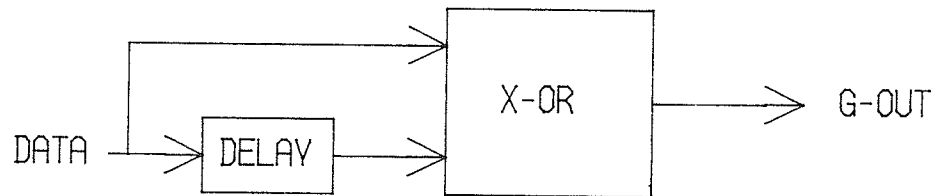
protocol and signalling scheme.

A delay element is generally needed between the DI module's ACK signal and whatever ACK is generated by a typical DI interface. The purpose of such a delay is to allow sequential logic into the local environment of the DI module. Normally the DI interface will produce an ACK signal upon satisfying all input and output conditions required by the ISG, and in fact that is exactly what the DI interface must do. However, we have sequential logic that is driven by the DI interface ACK signal. The local sequential logic needs some finite time to carry out its operations and update the sequential outputs. If we immediately sent out the ACK signal from the DI interface, we would allow the sender to possibly send new outputs before all sequential calculations were completed -- a clear violation of the DI restriction that all outputs must be updated prior to sending an ACK signal. Therefore, a delay must be introduced to ensure that all circuit outputs are appropriately updated prior to continuation of the protocol cycle. A logic delay that meets the worst case delay of the sequential logic must be provided in the simplest implementation, otherwise the logic must generate a completion signal using a more elaborate design. The latter configuration is preferable since it tends to optimize performance although it will almost always result in significantly more logic. In this manner the DI restrictions are preserved by the sequential logic in this circuit.

Unfortunately, introducing fixed delays is less optimal than having an entirely DI module. This is because the circuit would already have all the DI delays associated with it by generating a full logic function directly, while sequential logic must add an additional delay to assure correct interface operation. The principal advantage of using sequential logic is the fact that an ISG and EISG for the function need not be derived. Thus, many existing sequential cells in cell libraries may well be used with minor modifications. When one considers the man-years of development that such libraries entail this can be a significant advantage. Thus, a trade off between logic speed and design effort must be made.

Although assumptions regarding the internal configuration of the clocked logic circuit are not made, a few circuits would commonly be quite useful for interfacing sequential logic to the DI interface. First, we must remember that transitions on all outputs of a DI circuit are monotonic. Most clocked sequential logic, however, responds to a specific clock edge. One approach is to produce a pulse, or glitching, circuit that will produce both clock edges each time a full input cycle is completed. Figure 38 provides an example of just such a circuit. By introducing an appropriate delay in one input line of the exclusive or gate, one can produce a momentary pulse at its output. As we know, a sufficient delay will produce a lag in one of the edges reaching the gate. Momentarily, then, the gate will receive both '1' and '0' inputs which will produce a

(A) LOGIC PULSE CIRCUIT



X-OR EXCLUSIVE OR FUNCTION
G-OUT PULSE OUT

(B) TRUTH TABLE OF LOGIC PULSE CIRCUIT





| DATA | G-OUT |
|---|---|
| 1 | 0 |
| 0 | 0 |
|  |  |
|  |  |

Figure 38: Pulse Circuit

'1' at the output. Hence, a useful clock pulse would be generated.

A similar technique could be utilized on the data lines (DH/DL) to convert them to single rail binary data. Figure 39 shows a fairly simple method to convert the rail data into binary data acceptable to a typical sequential logic circuit. Although this technique utilizes a significant amount of logic, it has the utility of being highly robust and easily implemented using standard components. Note that it is desirable that DH and DL be fed directly into this circuit from the inputs and that a Macro-Module implementation of the DI interface only provide the ACK signal. In this manner, we may avoid some of the delay associated with the Macro-Module internal logic. However, this direct logic method is somewhat wasteful. The Macro-Module already provides a great deal of logic to handle the DL/DH signals; therefore, we are only using the input protocol for arbitration and timing.

Additionally, it may be necessary to convert binary data back to the standard DI double rail data convention. In such a situation, a simple circuit like the one shown in Figure 40 could be used. Finally, implementation of a central DI stage could result in a combination of the above conversion circuits and this is depicted in Figure 41. Remember that the general nature of the circuit implementation may greatly increase logic overhead; in many cases it would be desirable to implement this function within the sequential logic itself as part of the

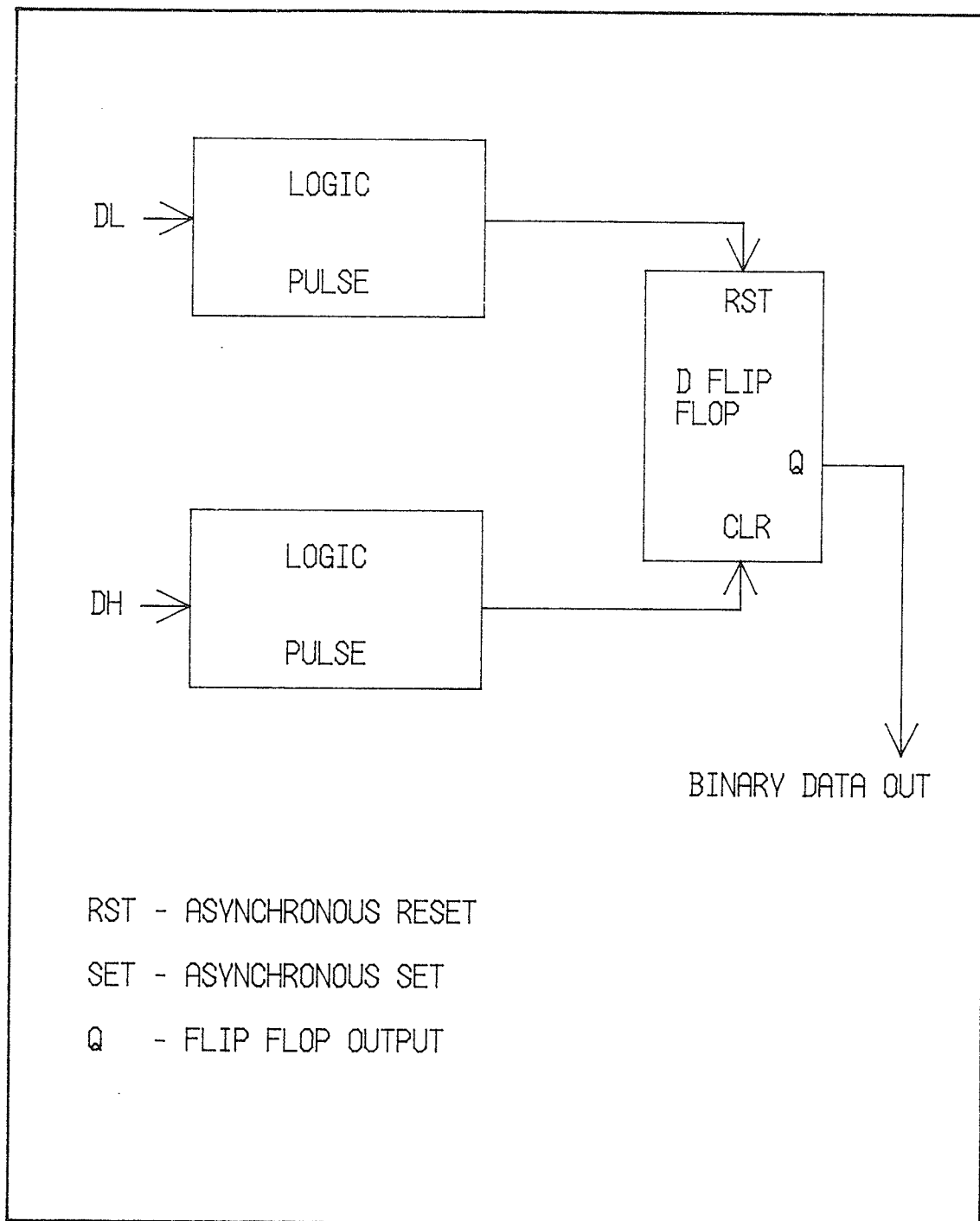


Figure 39: Rail Data to Binary

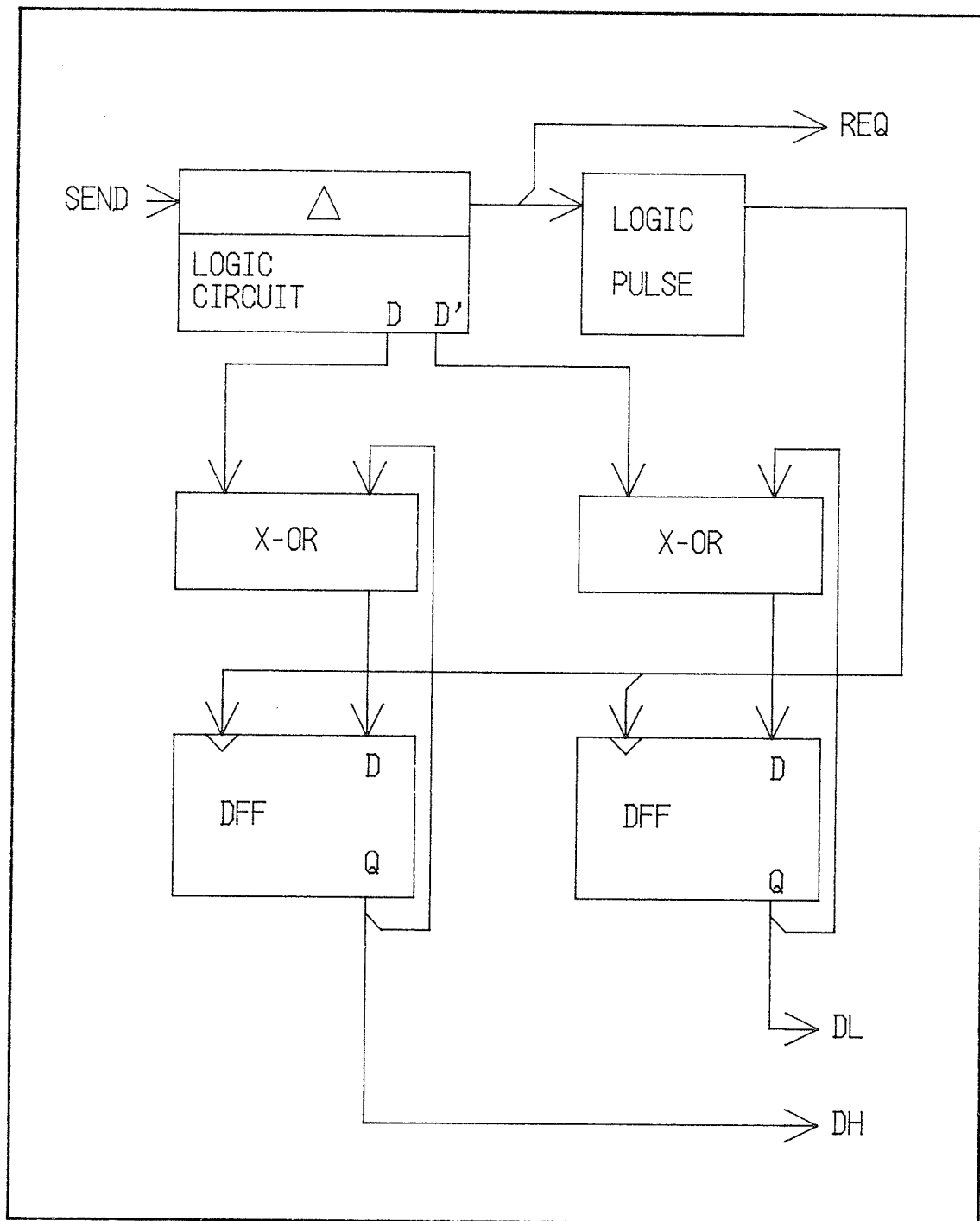


Figure 40: Binary to Rail Data

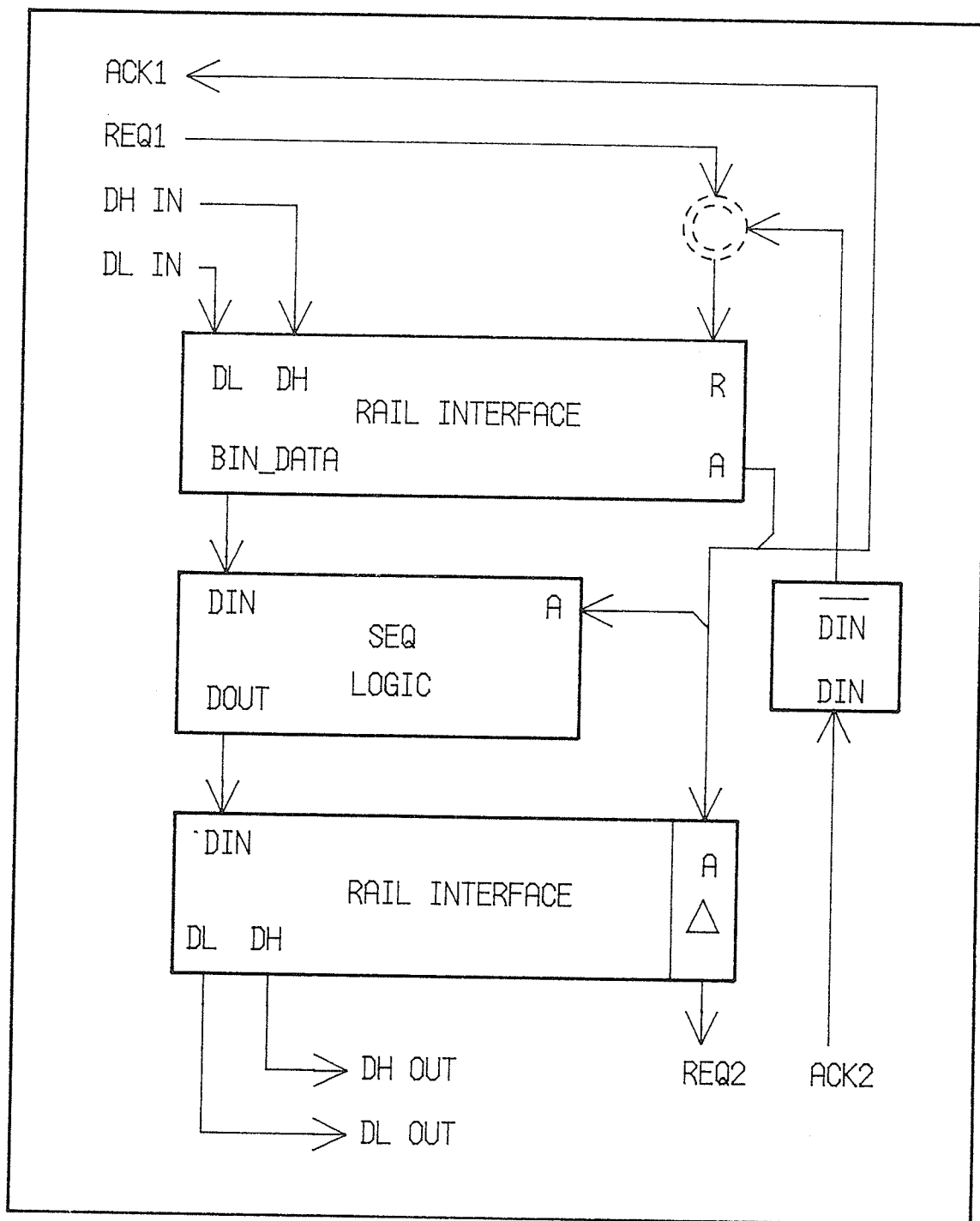


Figure 41: Central DI Stage

design process.

Ultimately, a judicious use of both fully DI modules and some mixed-mode circuitry may provide the best combination of design ease, efficiency, and flexibility. The next subsection discusses the system level and block level considerations in DI module usage.

4.3.3 Construction of a Driver Interface

It should be apparent from earlier discussions that a logic design involving many DI modules ultimately requires a driving circuit. Some module must initiate activity to start circuit execution. Further, it must continue to provide inputs or control as the circuit requires. Unfortunately, the task being implemented tends to shape the driver in almost all major respects. For example, in the ADIC communication chip previously described, I would probably use data writes by a processor as an initiating signal to begin data transmission. Data transmission would then continue until all remaining bits had been transmitted. Thereafter, another write would be needed to reinitiate activity by the sender in the communication link.

Essentially, all drivers require a number of basic elements. Figure 42 provides a typical driver circuit implemented in a mixed-mode format. First, the driver requires

a start-up or initiating signal to force activity. This signal prompts the computation or other activity that the circuit as a whole provides. Thereafter, initiation depends upon both the data involved, the arbitration and task the circuit must perform, and other application specific details. In some cases an initiating signal may be continuously needed during each cycle; in others, the circuit may be free-running depending upon dependent DI blocks. Thereafter, the driving and maintenance of circuit activity could simply depend upon received ACK signals, or a combination of external signals and ACK signals; the only limit is that the driver may not be allowed to supercede the restrictions of the DI protocol standard employed.

If desired, this cell could also be placed in a fully DI form. An ISG that will implement a DI version is very close in nature to a typical DI stage; the only major difference is the presence of an initiator or reset signal to force initial circuit operation. A simple ISG that implements a very basic driver is portrayed in Figure 43. Greater detail is difficult to provide due to the application-specific nature of circuit driving blocks.

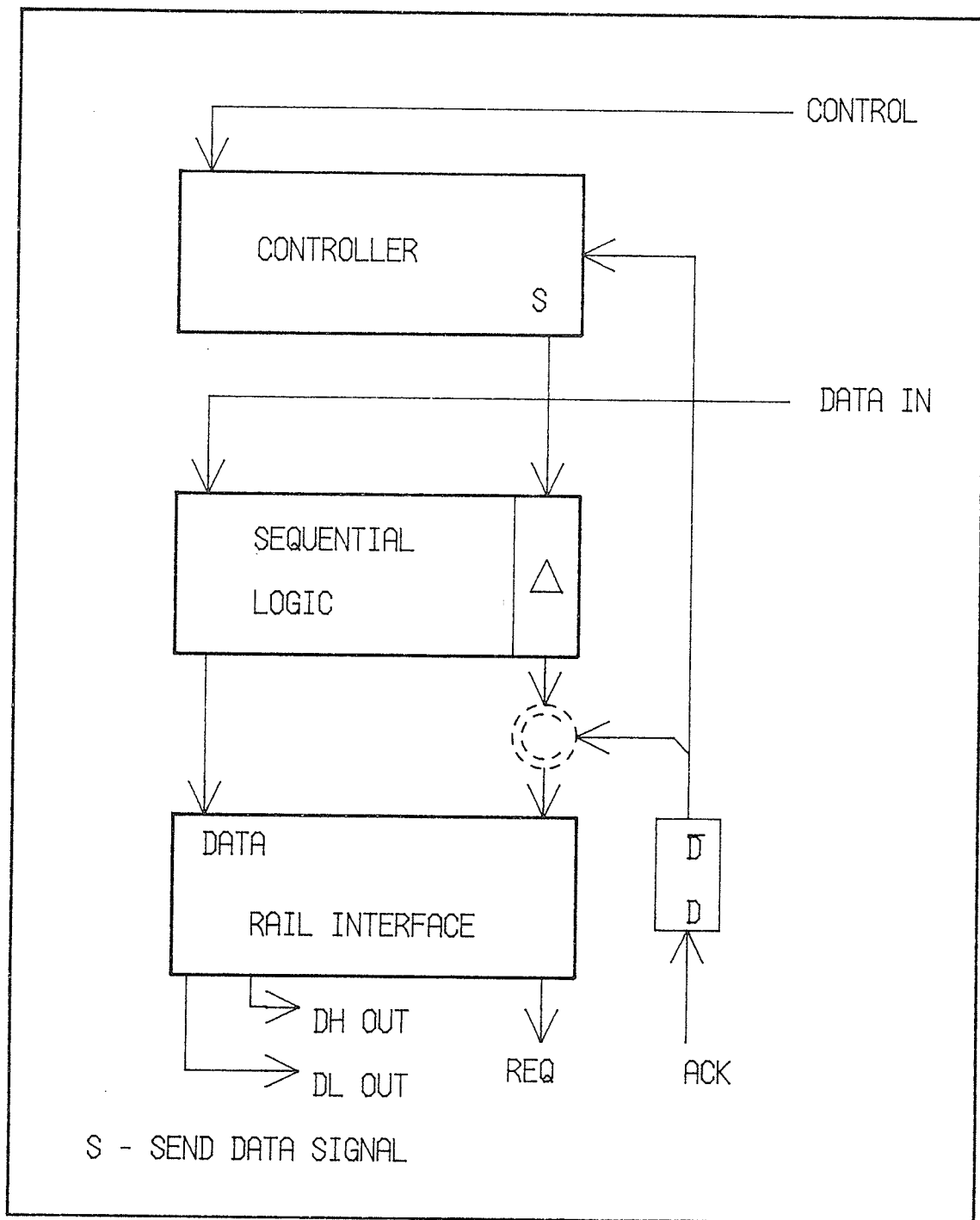


Figure 42: Driver DI Stage

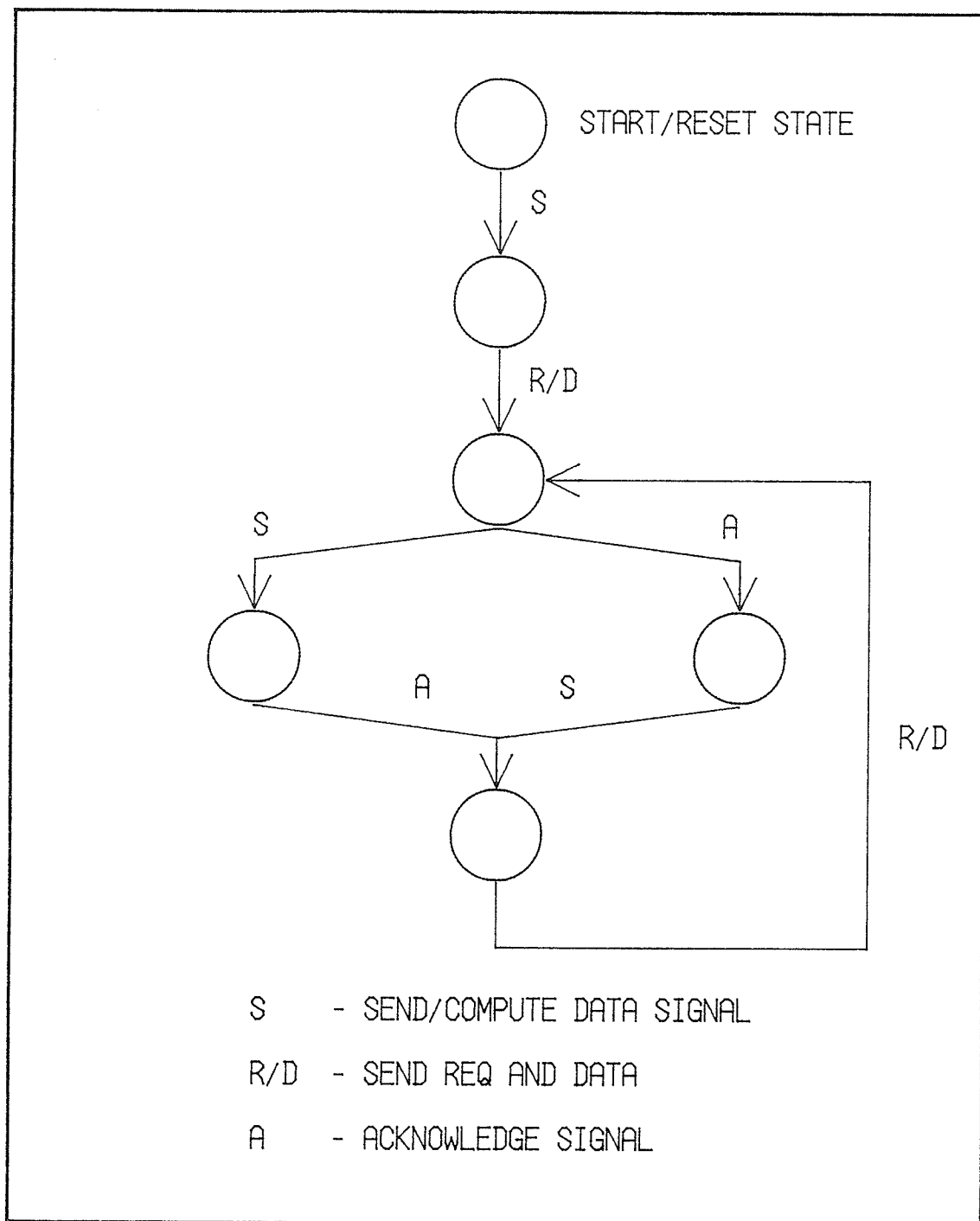


Figure 43: Driver ISG

4.3.4 System Level Design

A mixed mode DI module is capable of all the standard behavioral functions in a normal DI system. For example, a typical mixed mode DI module, such as the one pictured in Figure 41, implements a pipelined central stage. That stage may well have all of the features of the DI protocol, a sequential logic core, and input and output conversion circuits. Combining all the above with appropriate join functions produces a central pipelined stage as outlined previously.

Parallel and chain termination is produced in a similar manner. All a mixed mode method requires is the use of appropriate interface logic between the DI interface circuitry and the sequential and combinatorial core of the logic block. This methodology suggests that it may be convenient to utilize isochronic zones to accelerate design and testing in many DI systems.

Anyone curious about the proper interconnection of blocks using parallel, serial, pipelined or other methods should refer to chapter 3. The DI modules described here follow the protocol restrictions required for such configurations.

Is mixed mode or DI design excessively wasteful of IC area? I do not believe that the overhead need be prohibitive. For example, I estimate that I could convert a significant portion of my Asynchronous DI Channel IC to a fully DI block form with

only ten to twenty percent overhead. Admittedly, the IC has many aspects that make it amenable to such conversion, but it is still implemented in fully synchronous logic at the lower level. Further, a fully DI module may well be reasonably efficient. Some authors, such as Molnar and Fang [3][7], propose that fully DI cells may yield only modest overhead penalties from studies they carried out in producing a standard DI cell library (called MacroCells). Therefore, it is not necessarily true that excessive area overheads are the outcome of DI designs.

Further, the major advantage that must be considered is the greater design ease that DI modules provide to system level designers. The huge ICs of the future with one million or more transistors will be difficult to design in a purely synchronous manner. It may be far more essential to guarantee correct operation by design than to save ten or twenty percent in logic overhead. The problems that logic designers will face in avoiding critical edges, interblock delays, and other transmission problems between chips may well be prohibitive. DI modules, however, free the designer from many of these difficulties; thus, I believe that DI modules will provide a powerful tool in the design, construction, and reliability of the ULSI devices in the future.

The major short term challenge will be to provide DI structures and design methods amenable to major design efforts. In particular, DI structures that provide all of the basic gate

functions and common circuit elements would be highly desirable. Further, tools that simplify the logic derivation process from ISG are greatly needed. Additionally, more work on integrating sequential and combinational logic into DI modules and settings are warranted. In any case, a designer has an alternative to purely sequential design that may well prove to be the more desirable option freeing the designer from dependence on the clock.

CHAPTER 5: SUMMARY

Chapter 1 provides an overview of the synchronization problems increasingly found in IC and system designs. Since synchronization is consuming greater IC chip area and design effort, an alternate design methodology involving self-clocking could be attractive. Asynchronous logic could free many designs from the synchronization requirements that currently serve to limit many designs.

Chapter 2 details a class of circuits that exhibit Delay Insensitive behaviour. The task in Delay Insensitive design is reduced to providing a logical and correct interconnection between elements since such devices are immune to arbitrary line delay. Such a methodology should reduce the overhead and design complexity associated with clocking and synchronization within large circuits or systems. Finally, a Delay Insensitive protocol was introduced. This protocol could provide a consistent interconnection method between high-level logic within a system. The description of a Delay Insensitive Asynchronous Communication IC design is given in chapter 3. This IC provides two fully bidirectional eight bit serial DI communication links on one IC. It also implements bus interface logic to provide a memory mapped peripheral. Circuit

testing is enhanced by providing a simple scan path, and by allowing logic to drive internal states onto the IC external data bus. Analysis of the DIAC IC implementation demonstrated a number of areas where the design could be optimized to further improve layout and performance. Nonetheless, the Delay Insensitive IC overhead was deemed to be reasonable given the application. The IC is functional despite significant external driver problems, and it demonstrates a practical DI application.

Finally, Chapter 4 presents a number of purely DI representations of the input and output functions of the DIAC protocol which demonstrate the practicality of DI design methods. Additionally, a mixed mode design methodology permits a combination of traditional clocked circuits within a DI system framework. Mixed mode designs allow designers to quickly extend cell libraries to satisfy Delay Insensitivity by utilizing currently functioning synchronous elements. Additionally, designers can optimize circuit layouts by including a mix of fully Delay Insensitive cells and locally clocked cells. Further work could include the development of a full Delay Insensitive cell library implementing fundamental functions within that framework, and the redesign of the DIAC IC using these library elements. Additional work could provide software generation of EISGs from ISG circuits, and the automation of the circuit generation.

REFERENCES

- [1] A. Yakoviev, "Designing Self-Timed Systems", VLSI System Design, September 1985, pp. 70-83.
- [2] T.P. Fang and C. Molnar, Synthesis of Reliable Speed-Independent Circuit Modules, Technical Memorandum 298, Computer Systems laboratory internal Report, Computer Systems Laboratory Internal Report, Washington University, St. Louis, Missouri.
- [3] C. Molnar, F. Rosenberger, T.P. Fang, "Synthesis of Delay-Insensitive Modules", Proceedings of the 1985 Chapel Hill Conference on VLSI, Computer Science Press, 1985, Chapel Hill, pp. 67-86.
- [4] R. Miller, Switching Theory: Sequential Circuits, Vol. 2, John Wiley and Sons, 1965, New York, New York, Chapter 10.
- [5] S. Unger, Asynchronous Switching Circuits, Wiley Interscience, 1969, New York, New York, Chapter 8.
- [6] L. Conway and C. Mead, Introduction to VLSI Systems,

Addison-Wesley Publishing Company, 1980, Reading, Massachusetts, pp. 229-254.

- [7] C. Molnar and T.P. Fang, Preventing Problems Caused by Distribution of Delays in Clock-Free Realizations of Modules of Delay Insensitive Systems, Technical Memorandum 313, Computer Systems Laboratory Internal Report, Washington University, St. Louis, Missouri.

- [8] F. Rosenberger, C. Molnar, and T. Chaney, Q-Modules: Internally Clocked Delay-insensitive Modules, Technical Memorandum 312, Computer Systems Laboratory Internal Report, Washington University, St. Louis, Missouri.

- [9] K. Eshraghian and N. Weste, Principles of CMOS VLSI Design: A Systems Perspective, Addison-Wesley Publishing Company, 1985, Reading, Massachusetts, pp. 259-268.