A GATE ARRAY CHIP SET AS A FAULT-TOLERANT BUS INTERFACE UNIT BASED ON NUBUS PROTOCOLS

by

Kuo-yeang Tsai

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

J. G. Tront, Chairman

C. E. Nunnally

S. F. Midkiff

March, 1990

Blacksburg, Virginia

c. 2

LD
5655
V855
1990
T835
c. 2

A GATE ARRAY CHIP SET AS A FAULT-TOLERANT BUS INTERFACE UNIT BASED ON NUBUS PROTOCOLS

by

Kuo-yeang Tsai

J. G. Tront, Chairman

Electrical Engineering

(ABSTRACT)

Even with the performance of microprocessors expected to double within the next three to five years, the processing power increase offered by parallel processing has made multiprocessor systems very cost-effective. Each module in the multiprocessor systems will typically include a processor, coprocessor, cache, and main memory. This kind of architecture has generated the system-on-a-board distributed-intelligence concept, and the 32-bit multimaster buses thus come into play since these high-performance systems need to communicate with each other. During communication, commands and large blocks of data are transmitted across the bus. Along with the multiprocessor system, the single-CPU system continues to need a fast bus and wide data path to serve as a common I/O interface for terminals, disk storage devices, communication, and memory boards.

With the board size limited, the trend toward distributed intelligence increases the need to place more functions on a single board, and therefore bus interface unit (BIU) integrated circuits (ICs) play an important role in the design of new boards. Spaceborn systems must be fault-tolerant due to their high susceptibility to transient faults and the high costs of repair and maintenance. Hence, a gate array fault-tolerant bus-interface IC based on modified NuBus protocols is designed to meet these requirements.

The gate array IC design system HIGHLAND from United Technologies Microelectronics Center is used, along with other CAD tools such as the Berkeley VLSI Tool Set and LOGEN to generate

a layout for the BIU. Two programs are written to interface the necessary CAD tools. All the circuits are designed and simulated on a VAXstation 3200 (Ultrix-32) and VAX11/785 (VMS).

Acknowledgements

I would like to express my gratitude to my advisor Dr. J. G. Tront, for guiding me throughout the course of my graduate studies. I would also like to thank Dr. C. E. Nunnally and Dr. S. F. Midkiff for consenting to be on my committee amidst their busy schedule.

My thanks are also due to Mrs. S. E. Cline and Mr. Bob Lineberry for helping me throughout this project.

Last but not least, I express my deep sense of thanks to my loving parents and brothers for encouraging and helping me throughout my educational career and other pursuits.

Table of Contents

1. Introduction 1

1.1 Buses, Multiprocessor Systems, and Bus Interface ICs 1

1.2 Necessity of Fault-Tolerant Feature 2

1.3 Spectrum of Work Done 4

1.4 Organization of Thesis 6

2. The NuBus and the Fault Tolerant NuFTbus 7

2.1 Introduction to the Simple Cost-effective NuBus 7

2.1.1 Key Features of NuBus 8

2.1.2 Basic NuBus Structure 9

2.1.3 Protocol Requirements 12

2.1.4 Interrupt Operations 14

2.1.5 Arbitration 14

2.1.6 Bus Locking 15

2.2 Comparison Of NuBus With Other Buses 16

2.3 The Fault-Tolerant NuFTbus 19

2.3.1 ID Lines 19

2.3.2 Arbitration Logic 20
2.3.3 Clock/Reset Lines 20
2.3.4 Error Detection and Correction Coding 22

3. Overview of the Bus Interface Unit 24
3.1 Basic Operations of the BIU 24
3.1.1 Initialization 25
3.1.2 Master Side Operation 25
3.1.3 Slave Side Operation 25
3.2 Function Blocks of the BIU 26
BTC and NBC 26
Error Detection and Correction Units 28
Address Decode Units 29
Arbitration Unit 29
BIU Registers 29
Board and Bus Transceiver 30
3.3 Control Signal Names Between BIU And Board 30
3.4 Handshaking Signal Names Between BTC And NBC 33

4. IC Technology Selection 36
4.1 Programmable Logic Devices (PLD) 37
4.2 Full Custom Design 38
4.3 Cell-based Approach 39
4.4 Gate Array 40

5. Introduction to CAD Tools Used 43
5.1 Highland Design System 43
5.2 PEG 52

5.3 LOGEN 58
5.4 Static Hazards 69

6. Circuit Design and Simulation Results 70
6.1 A-Chip 71
6.1.1 Block Diagram of the A-Chip 71
6.1.2 Arbitration Module 71
6.1.3 Board Transfer Controller And NuFTbus Controller 80
6.1.4 Control Path 106
6.1.5 Bus Status Detection 110
6.1.6 Detection of Bus Locking 110
6.1.7 EDAC Sharing Interface 112
6.2 B-Chip 115
6.2.1 The Block Diagram of B-Chip 115
6.2.2 Data Path 118
6.2.3 Address Decode Unit 119
6.2.4 BIU Registers 125
6.2.5 32-bit EDAC Unit 127
6.2.6 Block Starting Address And Block Size Detection 128
6.2.7 Block Operation Address Incrementer 128
6.2.8 Block Size Decrementer 131
6.2.9 Timer to Wait Between Transfer in Block Mode 131
6.2.10 Retry Counting Circuit 134

7. Problems Encountered and Possible Future Work 137
7.1 Complexity, Speed and Structure 137
7.2 Testable Design For Gate Array ICs 140
7.2.1 Proposed Testable Design For The BIU 141

7.2.2 Arbitration Unit 142
7.2.3 BTC And NBC 142
7.3 Conclusion 142
7.4 Future work 145

Bibliography **147**

Vita **149**

List of Illustrations

Figure 1. Processors linked by a shared bus. 3
Figure 2. Spectrum of Work. 5
Figure 3. The major elements of a typical NuBus system. 11
Figure 4. Allocation of the NuBus address space. 13
Figure 5. Comparison of transfer rates [12]. 17
Figure 6. The voting circuit for clock/reset lines 21
Figure 7. The function blocks of the BIU. 27
Figure 8. The gate array principle. 42
Figure 9. Steps in implementing gate array ICs. 45
Figure 10. HIGHLAND design flow. 46
Figure 11. Header format of a McLDL program. 48
Figure 12. An example of a McLDL circuit description. 49
Figure 13. Circuit configuration for the McLDL example. 50
Figure 14. Example of SIL program. 53
Figure 15. McLDL modules to compare and verify. 54
Figure 16. Example of a PEG program describing a FSM. 56
Figure 17. State diagram for the PEG example. 57
Figure 18. PEG.SUMMARY file for the FSM. 59
Figure 19. Example of the LOGEN input syntax. 61
Figure 20. Input file to LOGEN derived from file PEG.SUMMARY. 64
Figure 21. Combinational logic in the FSM generated by LOGEN. 65

Figure 22. The complete finite state machine. 66
Figure 23. Gate level circuits for the FSM. 67
Figure 24. Block diagram of A-chip. 72
Figure 25. The arbitration module. 74
Figure 26. Simulation results for the arbitration module. 75
Figure 27. Subcircuit CGIA in the arbitrator. 76
Figure 28. Subcircuit ABARB in the arbitrator. 77
Figure 29. The arbitrator in the arbitration unit. 79
Figure 30. Hierarchical relations among the finite state machines. 81
Figure 31. Flowchart for state machine BTCA. 82
Figure 32. Flow chart for state machine BTCA (continued). 83
Figure 33. Flow chart for state machine BTCB. 84
Figure 34. Flowchart for state machine NBCRIA. 85
Figure 35. Flowchart for state machine NBCRIA. 86
Figure 36. Flowchart for state machine NBCRIB. 87
Figure 37. Master side single read operation. 90
Figure 38. Slave side single read operation. 92
Figure 39. Master side single write operation. 95
Figure 40. Slave side single write operation. 97
Figure 41. Master side block read operation. 99
Figure 42. Slave side block read operation. 102
Figure 43. Master side block write operation. 105
Figure 44. Slave side block write operation. 107
Figure 45. Control path. 109
Figure 46. Bus status detection. 111
Figure 47. Detection of bus locking. 113
Figure 48. EDAC sharing logic. 114
Figure 49. Block diagram of B-chip. 116

Figure 50. Block diagram of B-chip (continued). 117
Figure 51. Address decode unit. 122
Figure 52. Address decode unit (continued). 123
Figure 53. Simulation result for the address decoder. 124
Figure 54. Block diagram of the 32-bit EDAC unit. 129
Figure 55. Block starting address detection. 130
Figure 56. Block address incrementer. 132
Figure 57. Block size detection and decrementing. 133
Figure 58. Timer to wait between intermediate transfers. 135
Figure 59. Retry counting circuit. 136
Figure 60. Testable feature added to the arbitration unit. 143
Figure 61. Scan-set logic with parallel load. 144

List of Tables

Table 1. The NuBus signals. 10

Table 2. Measures of complexity. 18

Table 3. The NuFTbus signals. 23

1. Introduction

1.1 Buses, Multiprocessor Systems, and Bus Interface ICs

Open, standardized 32-bit buses are a relatively recent development. The prime reason for creation of such buses is the arrival of 32-bit microprocessors. Also important are the successes of open buses in the 8 and 16-bit worlds, advances in packaging, and a strong move toward multiprocessor systems. Therefore, the 32-bit buses are generally not just a wider data-path version of 16-bit buses, but are also known as **modern** or **advanced** buses because of improvements in arbitration, board addressing, interrupt signaling, and packaging. Therefore, with 32-bit multimaster buses and various powerful processors in open systems, the system-on-a-board distributed-intelligence concept has been generated; it is either a module in a multiprocessor system or a function block (such as a communication board) in a single-CPU system [1,2].

As for the multiprocessor systems, a shared bus is one of the popular topologies used to link processors together. This is shown in Figure 1 on page 3. Each processor in the system has its own local buses through which it accesses its own local memory and, possibly, I/O ports. For much of the time, the processors execute programs stored in their local memory and operate on their own data. When one processor wishes to communicate with another, it takes control of the

system bus through its bus interface unit (BIU). In this way it can access the resources of another processor by controlling the system bus. [3].

As for the bus interface circuitry in the BIU itself, the implementation with discrete ICs or PLAs can take up a substantial amount of board real estate since their arbitration scheme, burst-data-transfer features, or interrupt structure all add to the interface complexity. Therefore, bus interface ICs play an important role in the design of new boards. Using bus interface ICs, board designers can have the advantage that VLSI chips offer; these advantages are lower parts count, lower power consumption, and lower cost. Moreover, because of the extra board real estate that an interface chip affords, designers can add features and functions, even though the size of the board is fixed [4]. This is the motivation and objective of this project.

1.2 Necessity of Fault-Tolerant Feature

Reliability is of critical importance in situations where a computer malfunction could have catastrophic results. Examples include the NASA Space Shuttle, aircraft flight control systems, hospital patient monitors, and power system control. Since the cost of designing, building, and launching a space flight or a satellite for unmanned long-life applications is much too high to allow electronic failures to occur in space, the space-born systems not only should be functional, but must also have a high degree of fault tolerance [5].

Space-born systems are particularly susceptible to transient faults. Transient faults in space are generally caused by cosmic radiation, alpha particles, or other types of charged particles [6,7]. Transient faults cause no permanent damage but they can change the state of a flip-flop from 1 to 0 or vice versa. This is generally referred to as a single event upset or SEU. Hence an incorrect value in a flip-flop can make a finite state machine transit to unexpected states, change values of the flags used for some special purposes, or present incorrect data. Because many flip-flops are used in the

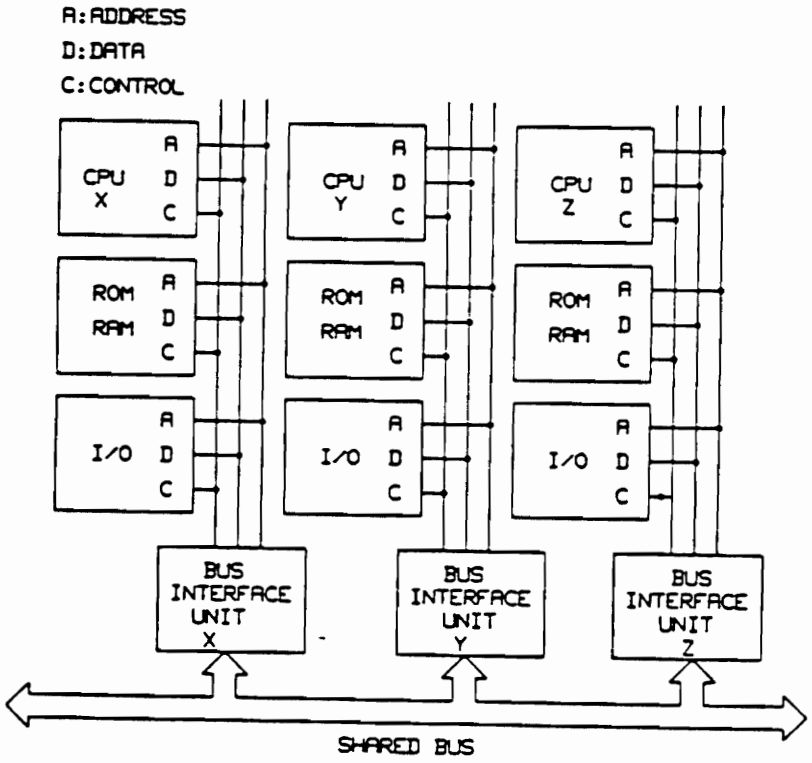


Figure 1. Processors linked by a shared bus.

BIU design, susceptibility of circuits to such upsets necessitates the incorporation of upset detection, location, and correction mechanisms in various subsystems as well as in the bus architecture.

1.3 Spectrum of Work Done

As shown in Figure 2 on page 5, the spectrum of work to be finished to design, fabricate, and test a gate array IC spreads roughly over nine steps as marked. The project to develop a fault-tolerant BIU has been finished to step 2 by Paranjape [14]. First, he documents the development of the NuFTbus, which stands for Nu (New) Fault Tolerant Bus, based on NuBus protocols. Then he develops the architecture of the NuFTbus BIU and the design of the function blocks in the BIU. He also developed a behavioral model for the controllers in the BIU by using VHDL (VHSIC Hardware Description Language) and simulated it to verify its functionality. Also, much simulation was done using the PCAD System on the bus arbitration unit under different conditions, with and without faults in it, to verify its function and prove that it is fault-tolerant. A PLA implementation of a portion of the BIU was also generated using the PEG (PLA Equation Generator) program and basing the PEG description on the behavioral model by Paranjape. My work for the thesis is a continuation of his work and focuses on a gate level implementation this corresponds roughly from step 2 to step 5 in Figure 2 on page 5. I also surveyed the CAD tools on the market and wrote programs that act as interfaces between CAD tools in order to make the CAD tool environment for this project as complete as possible. Based on the work done here and that of Paranjape, it will be possible to readily generate a physical realization of the BIU.

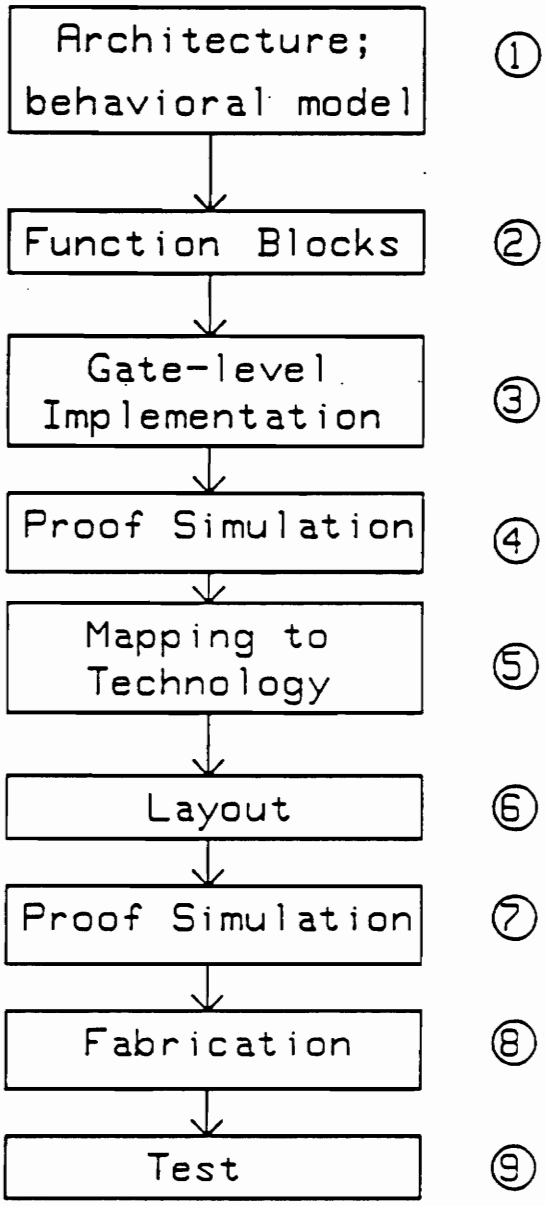


Figure 2. Spectrum of Work.

1.4 Organization of Thesis

This thesis is written to document the design and gate level implementation of the fault-tolerant version of the NuFTbus BIU. Chapter 2 makes an introduction to the simple cost-effective NuBus, it makes a comparison between various buses, and describes the fault-tolerant features found in the NuFTbus. Chapter 3 provides an overview of the BIU. Chapter 4 describes the IC technologies available to implement the BIU and presents the rationale for selecting a gate array. Chapter 5 briefly describes the CAD tools used during the design process, the Berkeley VLSI Tool Set, LOGEN, and the HIGHLAND Design System. Chapter 6 explains how each circuit module in the BIU works and shows the simulation results. Chapter 7 summarizes the problems encountered and outlines possible future work.

2. The NuBus and the Fault Tolerant NuFTbus

This chapter describes what NuBus is and makes a comparison with other buses. The materials in this regard are abstracted and summarized from [6, 8, 9, 10, 11, 12 and 14]. Later in this chapter, the rationale for incorporating fault tolerant features into the NuBus is presented and the fault tolerant features in the NuFTbus are explained.

2.1 Introduction to the Simple Cost-effective NuBus

The NuBus is an advanced 32-bit system bus that forms the backbone of several modern computer systems, including the NeXT Computer, Macintosh II, and the Explorer from Texas Instruments [6]. It is based upon the concept that simplicity is the most effective way to optimize the cost-performance point of the system design. We may say that a simple bus is analogous to a Reduced Instruction Set Computer. RISC computer architects believe that a computer's CPUs should be designed to do the frequent basic operations efficiently. The complex, less frequent operations which are more application specific, should be left out of the low level functionality. The NuBus takes the same position in the bus world.

The NuBus is uniquely suited for use in a wide range of computer and controller applications; the rugged construction of the pin and socket connector allows the NuBus to be used in harsh environments, such as industrial controllers and process control computers [8].

In minicomputer applications, the NuBus is used as the main system bus. All input/output (I/O) traffic is routed over the NuBus between the peripheral controllers and the system memory. Block transfer transaction are generally used to move large blocks of data from mass storage controllers to the system memory. If multiple processors are included in the system, data and commands are shared over the NuBus. Generally, it is not appropriate to use the system bus for instruction accessing, unless a cache is located on the processor. If all instruction accesses occur on the system bus, then the bandwidth available for I/O operations would be severely diminished [9].

In personal computer systems, the NuBus is used as the expansion bus to allow diverse speciality cards to be installed in the computer in order to tailer the capability of the system to a particular user's needs. The self-configuring and processor-independent features of the NuBus are especially attractive in this environment [10].

2.1.1 Key Features of NuBus

The NuBus requires only 51 signals as shown in Table 1 on page 10 [11]. This is only 19 more than the 32 needed for address and data. The simplicity, along with the synchronous nature of the NuBus, enhances system reliability and testability. This concept allows board designers to take full advantage of the bus's inherent performance without incurring the penalty of complex interface logic. The NuBus meets the following objectives:

1. System Architecture Independent
    - Optimized for 32-bit transfer, but 8-bit and 16-bit unjustified transfers are supported.
    - Not based on a particular microprocessor's control structure.

- Power up/down sequencing is specified by the system rather than dictated by the bus structure.

2. High Bandwidth
   - 10 MHz basic cycle time provides 37.5 mbyte bandwidth in block mode.
3. Simplicity
   - Reads and Writes are the only operations.
   - I/O and interrupts are memory mapped.
   - A simple service request feature is provided for boards that do not have bus master logic.
   - The single large physical address space allows all addressable units to be uniformly accessed.
4. Small Pin Count
   - Multiplexed address and data.
   - Only 51 signals, excluding power and ground.
5. Ease of System Configuration
   - ID lines provide geographical addressing and enable the system to be free of DIP switches and jumpers.
   - Distributed, parallel arbitration eliminates daisy chaining.

2.1.2 Basic NuBus Structure

The NuBus is a synchronous bus in that all signal transitions and sampling are synchronized to a central system clock. However, each transaction may be a variable number of clock periods long. This provides the flexibility of an asynchronous bus with the design simplicity of a synchronous bus. Figure 3 on page 11 shows the major elements of a typical NuBus system.

Only read and write operations in a single large address space are supported; I/O and interrupts are accomplished within these mechanisms. The modules attached to the NuBus are peers; no card or slot position is the default master or **special slot**. Each slot has an ID code hardwired into the backplane. This allows cards to differentiate themselves without jumpers or switches.

Table 1. The NuBus signals.

Classification	Signal	No. of Pins
Utility	RESET	1
	CLK	1
	PFW	1
Control	START	1
	ACK	1
	TM0	1
	TM1	1
Address/Data	AD<31..0>	32
Arbitration	ARB<3..0>	4
	RQST	1
	NMRQ	1
Parity	SP	1
	SPV	1
Slot ID	ID<3..0>	4
	Total Signals =	51
Power/Ground	+5	11
	-5	8
	+12	2
	-12	2
	GND	23
	Total Pin Count =	96

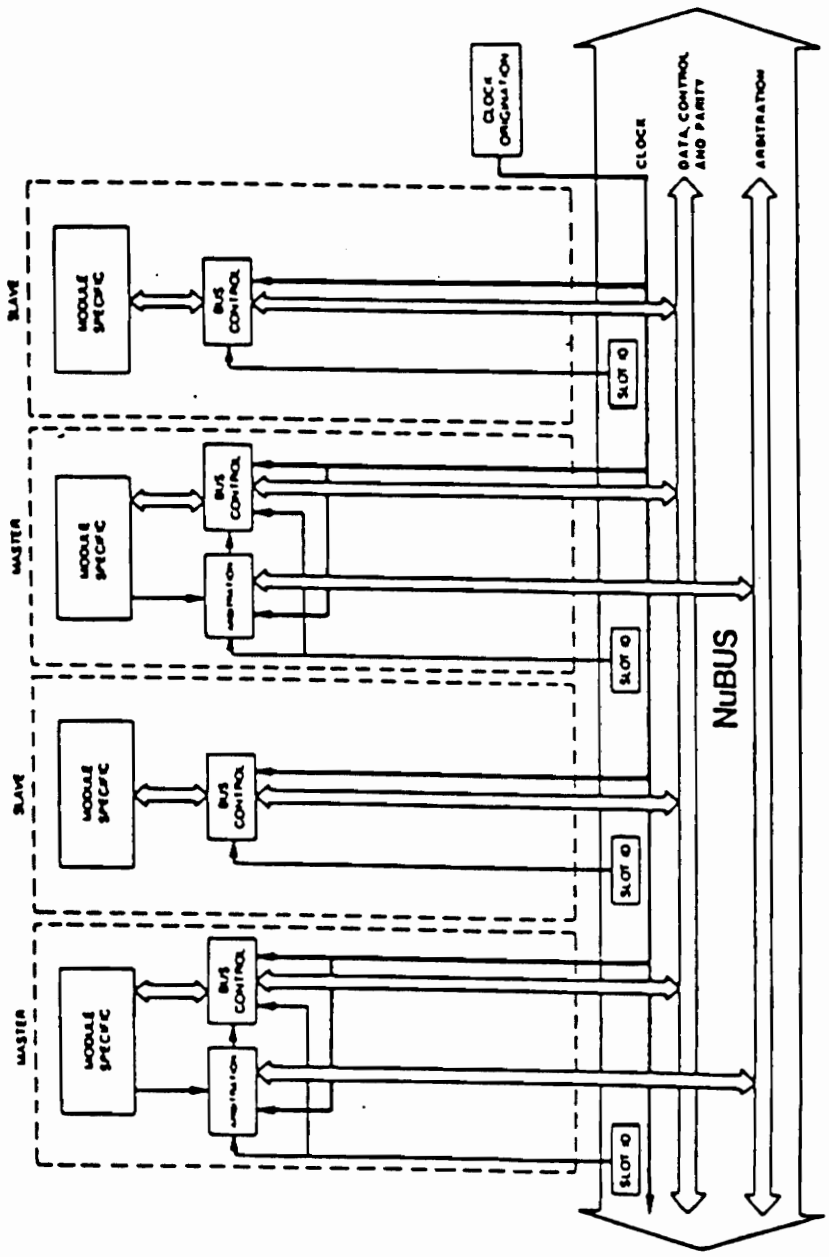


Figure 3. The major elements of a typical NuBus system.

A bus transfer consists of arbitration for access to the bus, transfer of the address and transfer of the data, and transfer of appropriate status information. Transfers may be 8-bit, 16-bit, or 32-bit transactions.

The NuBus supports multiple word block transfers of 2, 4, 8, or 16 words. The bandwidth of the bus is 37.5 mbytes/sec when block transfer is utilized.

The NuBus provides a linear address space of 4 gigabytes. The upper 1/16th (256 megabytes) of this address space is called **slot space**. As Figure 4 on page 13 shows, this area is further divided into sixteen, 16-megabyte pieces which are mapped to the 16 possible NuBus card slots. Addresses of the form F(ID)XXXXXX belong to a card's slot space where ID is a four bit slot identification code. This fixed address allocation, based solely on a board's slot location, enables the design of systems which are free of jumpers and switches. The remaining 15/16ths of the address space is uncommitted and allocated as required. Any allocation of that space is programmable via registers in slot space.

2.1.3 Protocol Requirements

The NuBus supports transactions of several different data sizes. Although optimized for transactions of words and blocks of words, the NuBus also support byte and halfword transactions. A byte of data is conveyed on the same signal lines regardless of the transaction mode. This unjustified data path approach allows strait forward connection of 8-bit, 16-bit and 32-bit devices.

A NuBus transaction consists of request made by a master and a response made by the addressed slave. Before a master can initiate a transaction, it must become the bus owner by winning an arbitration contest. Arbitration can occur in parallel with a transaction initiated by the prior bus owner.

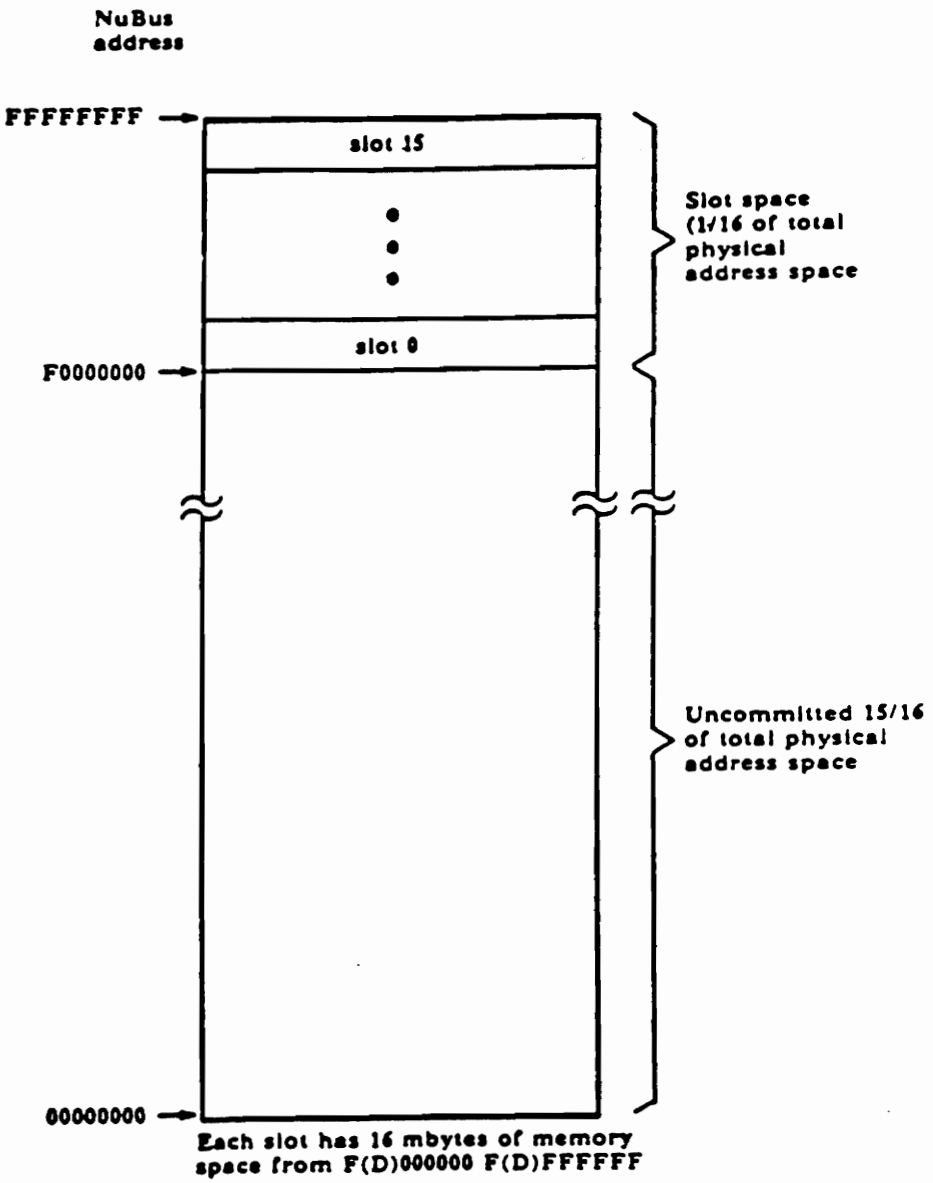


Figure 4. Allocation of the NuBus address space.

Transactions consist of single data cycle read or write transactions, or multiple data cycle read or write transactions known as block transfers. All operations on the NuBus are accomplished using only these two simple types of transactions within a single address space.

2.1.4 Interrupt Operations

Interrupts on the NuBus are implemented as write transactions. Interrupt operations require no unique signals or protocols. Any module on the NuBus can interrupt a processor module by performing a write operation into an area of memory that is monitored by that processor. Any address range on the processor module can be defined as its interrupt space.

For very simple modules that do not need the logic to become a bus master, but still need to request service, a simple service request mechanism is provided. Any module may drive the Non-Master Request (NMRQ) line which indicates that service is needed. This feature is generally used only in systems which use the desk top form-factor boards. In these systems, a larger board, called the **system board**, contains a portion of the system hardware, such as the processor and memory. Connectors on the system board allow NuBus cards to be plugged into the system. An individual NMRQ signal is connected from each NuBus connector to a priority encoder on the system board, thus allowing the identity of the module requesting service to be easily determined.

2.1.5 Arbitration

The NuBus fair arbitration mechanism differs from strict priority arbitration in that it prevents **starvation** of cards and distributes bus bandwidth evenly. This differs sharply from many buses that promote the idea of **priority**. Systems that employ priority based arbitration often exhibit strange and non-repeatable failures when one of the low priority boards is prevented from obtaining access to the bus before a buffer overflows or a timer times out. Often, rearranging the position of the cards

in the system causes the problem to go away, but one is never sure if the problem has been eliminated. Systems that use the NuBus are assured of a guaranteed maximum latency for every card in the systems. This allows adequate buffers to be designed into each card to avoid starvation problems.

2.1.6 Bus Locking

Although modules generally use the bus for short periods, sometimes a module must lock the bus. An example of this is an indivisible test-and-set operation performed in a multiprocessor environment. This requires the bus to implement **semaphores**, or signal bits, that are used to control the allocation of resources that may be shared by the various processors in the system.

Bus locking is accomplished with no added mechanism. To lock the bus, a master simply continues to request and contend for bus mastership. Since it has the highest ID code of those modules present, it wins subsequent arbitration.

In many cases, just locking the bus is not sufficient to implement an indivisible test-and-set function. Boards that have private buses that allow a processor to communicate to memory on that board must be able to prevent the processor from accessing its own local memory during the test-and-set sequence. In order to easily lock the internal bus of a NuBus module, such as a board with a processor and memory, a resource lock mechanism is provided. A one cycle transaction, called an Attention-Resource-Lock cycle, is broadcast to all bus modules by a master that intends to lock the bus. Any module that is addressed after this knows that locked transactions are occurring. At the completion of the locked sequence, another one cycle transaction, called an Attention-Null cycle is broadcast by the bus owner to indicate to all bus modules that the locked sequence is completed.

The NuBus has been adopted by the Institute of Electrical and Electronic Engineers (IEEE) as a recognized standard. A detailed specification of the NuBus is published as IEEE1196 [11].

2.2 Comparison Of NuBus With Other Buses

NuBus is an economical bus, yet it has higher performance than the STD Bus, STE bus, Multibus I and the IBMPC and AT buses. However, it is in the same economic class as these 8- and 16-bit buses. The maximum transfer rate comparison is shown in Figure 5 on page 17 [12].

NuBus's performance is roughly equivalent to that of Multibus II and VMEbus, although Multibus II , VMEbus and Futurebus are definitely in a different complexity class. This comparison is made in Table 2 on page 18.

NuBus, from the beginning, had the design goals of processor independence and multiprocessor support. In contrast, other buses like VME and Multibus were developed by chip makers, while PC and EISA buses have strong ties to the Intel family of microprocessors. Related to processor independence was a second criteria: architectural independence. Processor dependence is caused by details of specific signal timings, nonuniform address space, justification of data paths, etc., while architectural biases relate to presumptions about such things as whether systems will be tightly coupled or loosely coupled CPU's. NuBus used its simplicity goal to help achieve its processor and architectural independence goals. By taking the OSI (Open System Interconnection) communication model as a reference, the scope of the NuBus specification is restricted to the **link layer** protocol, any facilities such as message passing would be a convention above the bus specification level. Other buses, such as Multibus II, have explicit low level mechanism for message passing which are in addition to the basic bus hardware [13].

In a flexible multiprocessor system, resource availability issues like the number of processors present in a given computer, can make the initial booting up process quite complicated. NuBus boards have

BUS	Transfer Rate (bytes/second)
STDbus	2 M
PC	4 M
STEbus	5 M
ISA (PC/AT)	8 M
Multibus I	9 M
Micro Channel	20 M
EISA	33 M
NuBus	37 M
Multibus II	40 M
VMEbus	40 M

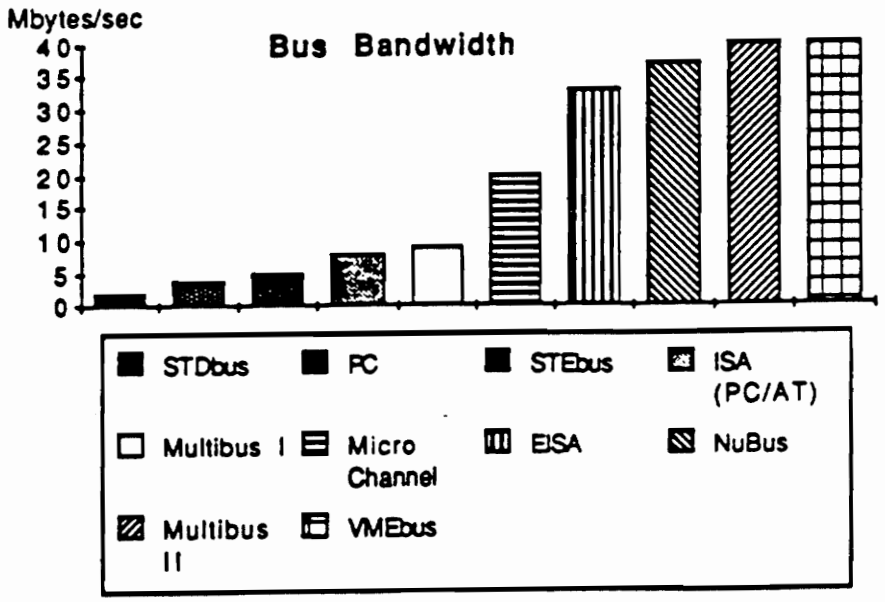


Figure 5. Comparison of transfer rates [12].

Table 2. Measures of complexity.

	NuBus	Futurebus	Multibus II	VMEbus
Number of:				
Address Spaces	1	1	4	18
Active Signal Lines	47	67	67	107
Interrupt Lines	0	0	0	7
Connectors Used	1	1	1	2
Transfer Types	4	8	6	8

a configuration ROM on each board which allows initialization software to detect what resources are available in a dynamic fashion on each power-up of a system. (The NuBus IEEE standardization committee is working with the IEEE Architecture Study Group to codify the structure of the contents of the configuration ROM) [9].

2.3 The Fault-Tolerant NuFTbus

If the backplane bus in a multiprocessing system is to exchange commands, messages or data between different processors, as stated in Chapter 1, it is necessary that the transfer on the bus be free from any kind of faults. The effects of faults is represented by means of a fault model. The type of fault models considered here are a stuck-at model and a temporary stuck-at fault model. It is assumed that there will be only a single bit failure at any given time. The NuBus is modified to make it fault-tolerant and is referred as NuFTbus (Nu Fault Tolerant Bus) [14]. Paranjape developed the fault-tolerant features in the BIU by adding a parity bit to the ID codes, duplicating the arbitration logic, using three lines for each of the bus clock and reset lines, using the error detection and correction units for the address/data lines and the control lines. These features are briefly explained in the sections follow. Details of these features and a figure of merit can be found in Paranjape's thesis [14]. Table 3 on page 23 shows a list of the NuFTbus signals.

2.3.1 ID Lines

The 4-bit ID codes ranging from 0000 to 1111 are used to identify the address range assigned to each module and are also used in the arbitration process. If a stuck-at fault occurs on one of the ID lines, this will leads to a bus error and would also lead to an address space overlap. To avoid the occurrence of these situations, a fifth bit (parity bit) is added to the ID codes. The addition of this parity bit gives the ID codes a distance of two. That is, any two codes differ by two bit locations. So, even

if one bit gets complemented, the code is still distinct from any other code by at least one bit position. These distinct ID codes prevent two or more modules from becoming the bus master at the same time.

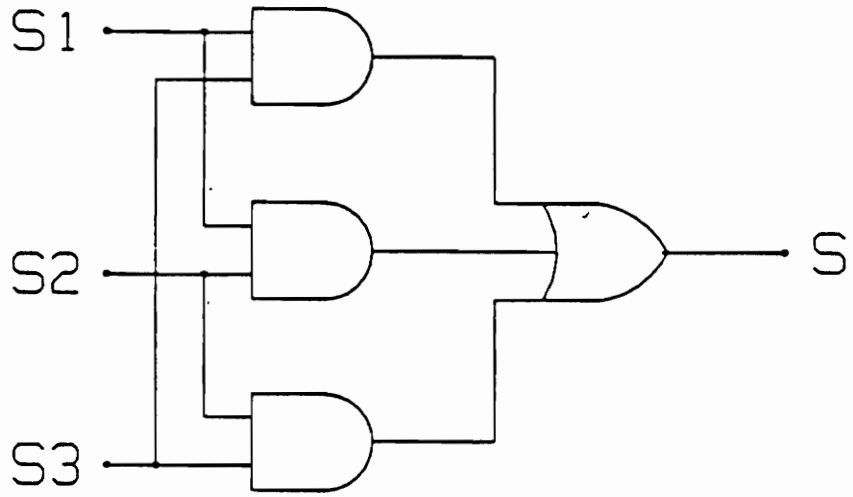
2.3.2 Arbitration Logic

Presence of a stuck-at fault on the arbitration lines (ARB_4- ARB_0) might lead to a system without a master or a system with more than one master. To get around these problems, two steps are taken. First, as explained earlier, a parity bit is added to the ID codes which gives the ID codes a distance of two. Thus, two modules can never win the arbitration contest at the same time. Secondly, the arbitration is carried out on two sets of arbitration lines, ARB_9-ARB_7-ARB_5-ARB_3-ARB_1 and ARB_8-ARB_6-ARB_4-ARB_2- ARB_0. Redundant arbitration takes place at each bit position.

Intermediate results at each ID bit are compared. If the ID bit is accepted at each redundant arbitration position, the arbitration is continued to the next bit position. If an ID bit loses at both ARB lines, that module stops contesting to become the bus master. If the two ARB lines corresponding to that ID bit do not match, that bit is bypassed and the arbitration is continued. So even if the module has a single bit error, it can arbitrate [14].

2.3.3 Clock/Reset Lines

The clock signal is made redundant and there are three clock lines on the bus. These clock lines are generated from the same oscillator and are voted in each BIU. A similar voting scheme is implemented for the reset line. The voting circuit in each BIU is shown in Figure 6 on page 21.



S1	S2	S3	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 6. The voting circuit for clock/reset lines

2.3.4 Error Detection and Correction Coding

Error detection and correction coding using a modified Hamming code is carried out on the address/data and the control lines of the NuFTbus. Check bits are added to these lines. Single bit error correction and double bit error detection requires the minimum distance between code to be 4, so a modified Hamming code is used [15]. For a 32-bit address/data path, seven check bits are added. Four check bits are added to the control signals START, ACK, TM_1 and TM_0. These check bits can detect and correct single bit errors and can detect double bit errors.

Table 3. The NuFTbus signals.

RESET < 2:0 >	initializes all the modules to power-up state
CLK < 2:0 >	10 MHz clock
PFW	power fail warning
ID < 4:0 >	ID lines
NMRQ	non-master request
AD < 31:0 >	multipexed address/data lines
AD < 38:32 >	check bits on AD < 31:0 >
TM < 1:0 >	control and status lines
START	indicates the start of transaction
ACK	indicates the ending of a transaction
CNTL < 3:0 >	check bits for control signals
RQST	indicates that the module would contest for bus
ARB < 9:0 >	wire-ORed arbitration lines

3. Overview of the Bus Interface Unit

The architecture of the NuFTbus Bus Interface Unit (BIU), the functional block configuration and the names and functions of all the signals, was reported by Paranjape in [14]. The purpose of this chapter is to provide an overview of the NuFTbus BIU. Basic operations of the BIU are described first in section 3.1. Section 3.2 succinctly explains each function block in the BIU. The board-BIU interface and BTC-NBC handshaking are explained later in the chapter, where BTC stands for Board Transfer Controller and NBC stands for NuFTbus Controller.

3.1 Basic Operations of the BIU

The module which wins the arbitration contest and gains control of the bus so as to initiate a NuFTbus transaction is referred to as a master module. An operation carried out by a bus master's BIU is called a master side operation. Similarly, the addressed module which responds to the master module is referred to as a slave module and the corresponding operation carried out by its BIU is called a slave side operation.

3.1.1 Initialization

Among those modules connected to the NuFTbus through their bus interface units (BIUs), there is one module designated as the SYSTEM TEST/Boot Master (STBM) module. Upon power up, the STBM resets all of the BIUs and loads initial control information into the command register of all BIUs. Once the bootup is finished, if any module has a processor on board, that processor can load control information into the command register in its own BIU. After all of the required control information, such as the memory start address and the size of the global address space allocated to that module, and the number of retries for a failed operation, has been initialized, the BIU can initiate or accept a bus transaction on the NuFTbus.

3.1.2 Master Side Operation

Whenever the processor on a board issues an address on its local bus, the BIU latches that address along with the necessary control signals. The address is decoded and if it is in the address space implemented on the board itself, the BIU does nothing further. If the address is within the memory space implemented on some other module, the BIU decodes the control signals and determines the type of operation to be carried out. The operation can be a single word read, single word write, block read or block write. Next, the BIU outputs a wait signal to the processor. The BIU then takes part in the arbitration contest to become the bus master. Once the BIU obtains control of the bus, a NuFTbus transaction is started. With an acknowledgement from the slave BIU, the wait signal being sent to the processor is deasserted indicating that the operation is complete.

3.1.3 Slave Side Operation

Now consider the operation of the BIU from the slave side of the bus. When the BIU sees a low on the START signal line with ACK deasserted on the NuFTbus, it latches the value on the ad-

dress line and decodes it. If the address is in this particular BIU's address space, then the BIU completes the bus transaction as a slave and sends back an acknowledgement to the bus master BIU. In the slave mode, the BIU requests the local processor to release the local bus and it completes the transaction without involving the on-board processor if the transaction is not an interrupt to the processor.

3.2 Function Blocks of the BIU

The function block diagram of the BIU is shown in Figure 7 on page 27. The function blocks are listed below and are followed by sections which explain how they perform in the BIU.

1. Board Transfer Controller (BTC)
2. NuFTbus Controller (NBC)
3. Error Detection and Correction Unit (EDAC)
4. Address Decode Unit
5. Arbitration Unit
6. Address, Data, Control and Status Registers
7. Board and Bus Transceiver

BTC and NBC

Board Transfer Controller (BTC) consists of two finite state machines: BTCA and BTCB. It performs the handshake operations between the processor or the control logic on the board and the BIU. NuFTbus Controller (NBC) consists of four finite state machines; they are NBCRIA, NBCRIB, NBCRB and NBCFB. It controls the arbitration logic and performs the interface operation with the NuFTbus. The different tasks of the BTC and the NBC are as follows:

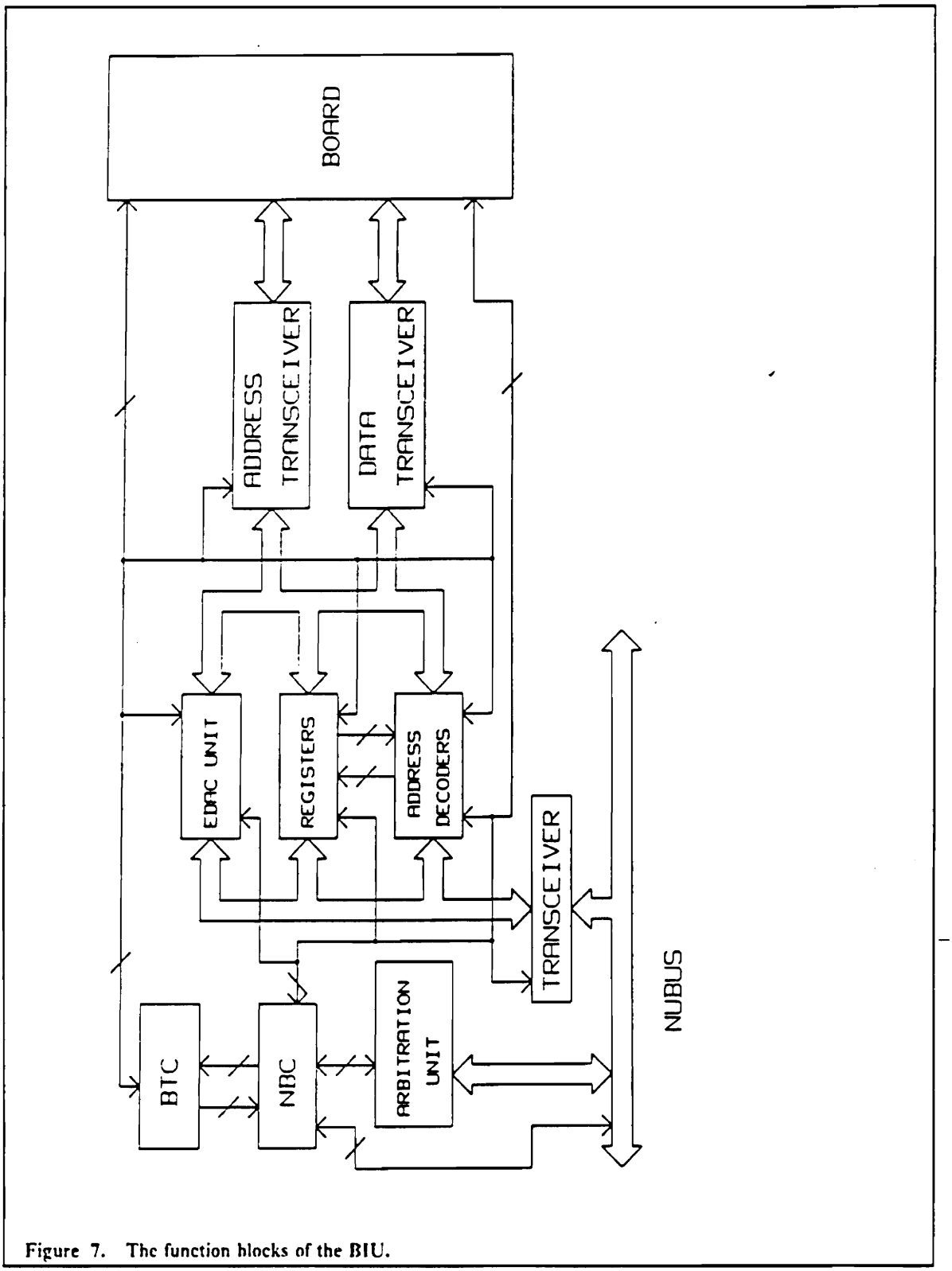


Figure 7. The function blocks of the BIU.

- BTC is used
  1. to decode the address from the board
  2. to generate/check EDAC on the address and data lines
  3. to indicate to the processor double errors in the address or data lines and to set corresponding flags in the BIU status register
  4. to receive the data word from the NBC and pass it to the processor
  5. to complete the BIU register read/write operation in master mode
  6. to put the master mode transaction on hold if a slave mode transaction is already started by the NBC

- NBC is used
  1. to arbitrate to be a bus master
  2. to start a bus transaction for read/write operations
  3. to generate EDAC on control signals
  4. to receive data from a slave on the NuFTbus, check EDAC and pass data to the BTC
  5. to monitor the NuFTbus for the bus status (busy or available)
  6. to decode the address on the NuFTbus and respond if the BIU is being addressed
  7. to put the on-board processor in the hold state by raising REL to complete a slave mode transaction
  8. to retry a failed operation
  9. to monitor arbitration errors
  10. to decrement block size and increment block address in block transfer
  11. to wait for a specified number of cycles between transfers in the block mode
  12. to set the error flags in the BIU status register and to inform the processor about the error
  13. to complete a BIU read/write operation in the slave mode

Error Detection and Correction Units

There are two error detection and correction (EDAC) units in the BIU. A 32-bit EDAC is used for the address/data lines and a 4-bit EDAC is used for the control lines. The EDAC unit for the address/data is shared by the BTC and NBC. When starting a transaction, the BIU on the master

side generates or checks on the check bits of the control and address lines, and also on the data if it is a write operation. The BIU on the slave side verifies the check bits for validity of the data. The BIU corrects single bit errors and flags double bits errors. After the transaction on the slave module is finished, the slave BIU generates the check bits for the status lines, and for the data if it is a read operation. The master BIU checks the validity of the status lines for the data word and takes action according to the EDAC results.

Address Decode Units

Since the BTC and the NBC have to be capable of working independently to allow for overlap of master and slave side operations, two address decode units are used; one for the address received from the board side and one for the address received from the NuFTbus side. The address decode units indicate whether the address is in the BIU space, slot space, or global space. If it is not in one of these three, then the BIU determines that it is in the memory space in some other module.

Arbitration Unit

Since NuFTbus uses a distributed arbitration scheme, each BIU has an arbitration unit to contest for the bus mastership [14].

BIU Registers

Each BIU has eight registers.

1. Global Memory Starting Address Register (32 bits)
2. Board Status Register (32 bits)
3. BIU Command Register (32 bits)

4. BIU Status Register (32 bits)
5. Data In Register (39 bits)
6. Data Out Register (39 bits)
7. Address In Register (39 bits)
8. Address Out Register (39 bits)

Board and Bus Transceiver

With the separate data bus and address bus in the board, the BIU has separate data and address transceivers in the BIU-board interface. On the NuFTbus side, a 39-bit multiplexed address/data transceiver/latch is used.

3.3 Control Signal Names Between BIU And Board

1. **SENI**: Single Enable In is an input to the BIU. A negative edge on SENI indicates the start of a master operation. The BIU latches address and data with this edge. Once the operation is completed, SENI goes high after which the BIU goes to its initial state.

2. **SENO**: Single Enable Out is an output from the BIU. SENO functions in the same way as SENI but it is driven by the BIU instead of by the board.

3. **BLKEN**: Block Enable is an input signal to BIU. A negative edge on BLKEN along with a negative going edge on SENI indicates the start of a block operation. BLKEN goes high when the intermediate word in a block is transferred. BLKEN goes low again to start the next word transfer. After the last word of the block has been transferred, both SENI and BLKEN go high simultaneously to indicate the end of the block operation.

4. **RWBI**: Read/Write In is an input to the BIU. A low on it indicates a write operation while a high indicates a read operation. (RWBI corresponds to TM_1 signal on the NuFTbus.)

5. **RWBO**: Read/Write Out is an output from the BIU. It is used for reading from or writing to the on-board memory.

6. **BYTE/OTHER**: this is an input to the BIU and corresponds to the TM_0 signal on the NuFTbus. It indicates whether the operation is on a byte, halfword or word of data or if it is a block operation.

7. **AEPR**: Address EDAC Provided is an input signal to the BIU. A high on AEPR indicates that the on-board memory has implemented the EDAC scheme and seven check bits are provided along with the address. A low on AEPR indicates that the check bits are not provided and the BIU should generate them.

8. **DEPR**: Data EDAC Provided is an input signal. It functions in the same way as AEPR.

9. **WAITI**: Wait In is an input signal. It is used in the slave mode when the BIU writes to the on-board memory. If a memory is slow, the memory makes WAITI goes high for the duration of the operation. Thus, if the memory is fast, it need not do anything to the WAITI line. Also the BIU need not see a pulse on the WAITI line. This saves having to handshake with the fast memories and still allows for slower memories to be used.

10. **WAITO**: WAITO is an output. It is used when the board is in the master mode. With the start of a transaction, the BIU makes WAITO high. It remains high until the transaction is over. In the case of a read operation, the master processor can latch the incoming data with WATIO going low.

11. **RMW**: Read Modify Write is an output signal. When the BIU does a byte or a halfword write operation, it first reads the contents of the address where the byte or halfword is to be written.

Then the BIU modifies the particular byte or halfword, generates check bits on the modified word and writes the whole word back. RMW is held high during this operation.

12. **REL**: Release is an output. Whenever the BIU is in the slave mode, it requests control of the local bus from the processor by raising the REL high.

13. **RELACK**: Release Acknowledgement is an input. A high on RELACK indicates that the BIU can take control of the local bus.

14. **BUSY**: This is an input. A high on this signal indicates that the local bus cannot be released to the BIU. The BIU will generate a Try Again Later acknowledgement to the master BIU.

15. **LOCKBUS**: Lock Bus is an input. Whenever a processor wants to do two or more consecutive transactions on the NuFTbus, but not a block transfer, it sets the LOCKBUS high. With the LOCKBUS input high, the BIU does not release control of the NuFTbus after the first transaction. Rather, the BIU retains control of the bus until the LOCKBUS is low. This signal is used when the processor does a "Test and Set" operation on a memory location. However, there is limitation on the number of transactions that can be carried out in the bus lock mode. If a processor tries to lock the bus for a number of cycles exceeding the pre-programmed maximum allowable number, then another BIU that is waiting for the bus may generate an attention cycle to restart bus arbitration.

16. **BIU/SYSTEM RESET**: It is an active high input. This signal is generated by a processor to reset its BIU or the NuFTbus.

17. **ERROR**: It is an output. ERROR indicates that an error occurred during a NuFTbus transaction. The error may be a double bit error detected by the EDAC units, a failure of all retry operations, or an error/timeout acknowledgement from the slave. The error status is written to the BIU status register and can be read by the processor.

18. **CLK**: CLK is an input from the board and is used for internal operation of the BIU. The maximum clock frequency depends on how fast the BIU can run.

3.4 Handshaking Signal Names Between BTC And NBC

The two finite state machines BTC and NBC interact through internal handshaking signals. Those signal are defined and listed below.

The following signals are driven by the BTC:

1. **RWNO**: A high on Read/Write NuFTbus Out indicates a master read operation and a low indicates a master write operation.

2. **TOR**: In the master mode, a high on the Transfer Out Ready tells the NBC that it can start the bus mastership arbitration followed by a corresponding NuFTbus transaction. In the slave mode, a high on the TOR tells the NBC that the operation has been completed by the BTC, and the NBC can send back an acknowledgement.

3. **RFD**: When Ready For Data is high, it indicates that the BTC is waiting for data from the NBC.

4. **NLOCK**: A high on NBC Lock indicates that the BTC is ready to pass a master operation to the NBC.

5. **BACK**: BTC Ack is a produced signal in response to the BTC Lock (BLOCK) from the NBC.

The following signals are driven by the NBC:

1. **RWNI**: A high on Read/Write NuFTbus indicates a slave read operation and a low corresponds to a slave write operation.

2. **TIP**: A high on Transfer In Progress indicates that the NBC is currently carrying out a transaction. In the master mode, TIP goes high when TOR is high and the NBC is ready to start the transaction. In the slave mode, TIP goes high after NBC receives RELACK from the board.

3. **AIR**: In the master mode, Address In Ready going low indicates to the BTC that the transaction was completed. In the slave mode, AIR going high indicates that the NBC has a valid address ready from the NuFTbus for a read or write transaction.

4. **DINRDY**: Data In Ready is used in the slave write mode or master read mode when the data from the NuFTbus is available for the BTC.

5. **BLOCK**: BTC Lock is used to tell BTC that NBC is ready to pass a slave operation to it.

6. **NACK**: NBC Ack is a signal in response to NBC Lock from BTC.

7. **SBLKRD**: A high indicates that the slave operation is a block read operation.

8. **SBLKWR**: A high indicates that the slave operation is a block write operation.

The signals listed above are either signals between the board and the controllers in the BIU or signals internal to the controllers. Those signals generated by the controllers to control other circuit

modules in the BIU and the input signals to the controllers generated by other circuit modules are not listed here. They can be found in the PEG programs in which they are documented.

4. IC Technology Selection

Whenever manufacturers develop a new piece of electronic equipment, they should consider at some point whether some all of the electronics should be integrated onto one or more silicon IC. The choice is thus whether to implement the design using off-the-shelf SSI (small scale integrated circuit) and MSI (medium scale integrated circuit) chips, or using a full custom or semi-custom technique [16]. The semi-custom technique can be further divided into three categories: Programmable Devices, Cell-Based Systems and Gate Arrays. As for the SSI/MSI-technique, it is ruled out here because of the rationale for the BIU design presented in Chapter 1.

Although the choice between technologies is based on the customer's own applications and priorities, in this chapter, the features of the technologies considered will be briefly introduced, and some advantages and disadvantages of each technology will be presented according to to the following four factors:

- total cost (development cost, unit cost, re-design cost, etc.)
- timescale (time from initial design to production of chips)
- design technique capability and flexibility
- ease and accuracy of design to first success (no debugging and re-design is needed).

4.1 Programmable Logic Devices (PLD)

The Programmable Logic Array (PLA) and Programmable Read Only Memory (PROM) are the best-known examples of programmable logic devices [17]. They may be regarded as lying half way between a standard product and a custom/semi-custom part. All such devices are standard production products up to the final stage of commitment to a particular custom requirement. This final commitment is usually arranged either by making certain final connections on the chip (mask-programmable), or by destroying connections already made on the chip (field-programmable).

Although one of the oldest semi-custom techniques, PLD is confined to a rather small area of application due to its inflexibility.

1. Cost - Development cost is very low, a few man of effort to construct the program and the use of an inexpensive programmer. In production the chips are only a few dollars each, though the unit cost should be compared on a cost/logic function basis.

2. Timescales - Development timescales are a few days and often only hours are needed to correct a design flaw. In production a mask-programmed device can be produced in large volume.

3. Flexibility - Only a very limited range of functions are available.

4. Ease and accuracy - Due to its limited capability, development is normally a straightforward job and first time success is common.

In general, programmable devices are quick and cheap to develop and produce, but they have a very limited application range due to small amount of logic a chip can offer, especially when compared with gate arrays.

4.2 Full Custom Design

In a full custom design approach, the objective is to produce the most efficient final design-on-silicon within given time or cost constraints. That is, to make the circuit occupy the smallest chip area, and/or have the highest speed performance. Usually, this is done by persons who are expert in detailed design of gate structures and interconnect topology at the silicon level. The following list presents the features of a full custom design:

- Incorporating only circuitry which is functionally necessary, with no unused gates or I/O buffers.
- Possible use of different device geometry sizes in different parts of the circuit for speed or area consideration.
- Interactive CAD/designer placement/routing of the chip.
- Possible introduction of novel gate structures to provide specific combinational logic functions.

When compared with gate array technology, this technology has the following attributes:

1. Cost and Timescale - The overriding feature of the full custom is the long and hence costly manual design phase. Several years of effort are normally expended for a fully hand-crafted design. Fabrication costs are almost the same as those of cell-based technology but higher than those of gate array.

2. Flexibility - This is the only type of silicon integration which is totally flexible.

3. Ease and accuracy - Because of its total flexibility, CAD tools tend to be large and complex and therefore usually lead to a higher probability of error.

The prime aim of full custom design is usually to achieve minimum chip size. However, to most of the equipment manufacturers, the fast access to of prototypes is of key importance in order to be able to introduce new products to the market as early as possible. Hence full costume design technology is not suitable for this purpose.

4.3 Cell-based Approach

The prerequisite of the cell-based approach is that there must already be available a set of proven standard circuit designs such as gates, latches, and other functional entities. These cells are custom-designed and held in a cell library. The procedure for a semi-custom cell-based design is listed below.

- Partition the customer's requirements into functional macros corresponding to available cells in the library.
- Place the required cells and design their interconnections.
- Simulate the complete circuit to check for errors.
- Obtain customer's assurance that the design meets the requirements.
- Generate the complete set of masks and fabricate the circuit in the conventional manner.

This is the main rival to gate array technology and is compared with gate arrays according to the factors listed below.

1. Cost - The use of automated layout means that the design portion of the development cost will be dominated by the circuit simulation costs and will be similar to a gate array design cost. Fabrication costs to produce prototypes will be considerably larger than the gate array costs as a full mask set must be produced and processed.

2. Timescales - Development timescales are almost the same as with gate array, as discussed in the cost comparison. As all masks must be made and all stages processed, fabrication times to prototype are longer than for gate arrays.

3. Flexibility - The cell-based system is more flexible than the gate array since it can use pre-designed cells for any function. But, it does not have the total flexibility of full-custom design because of the problems in maintaining the cell library.

4. Ease and accuracy - Since it uses almost the same kind of CAD programs for simulation, layout and testability analysis as gate array technology does, there is no big difference between them in this factor.

Generally, cell-based approaches are more suited to higher volume applications than gate arrays.

4.4 Gate Array

A gate array is a chip containing a regular matrix of logic gates or components which is pre-processed up to the final stages of metal layer patterning. Approximately 90 percent of the wafer processing steps have been completed before the chips reach the stage where they will be committed to a particular function. The individual gates or components can be connected together in a unique way to realize any required function, usually with only one masking step. The principle is illustrated in Figure 8 on page 42.

At this time, the largest sector of the semi-custom market is that of the gate array. They are currently available in all modern circuit technologies, such as CMOS gate arrays, ECL gate arrays and GaAs gate arrays.

1. Cost - The development cost of a gate array is the sum of the design costs and the prototype fabrication costs. The design costs will depend on the size of the circuit and array, the circuit complexity and the design aids available.

2. Timescales - With a good, well-proven design, translation to gate array logic and simulation can be done in weeks. However, if the design has hazards or is badly thought out, it could take months to modify it into a form suitable for a gate array. As a rule of thumb, six months is a realistic development time.

3. Flexibility - Gate arrays can accommodate virtually any purely digital function on one chip, except for the memory components (since it is too wasteful in gate array area).

4. Ease and accuracy - The gate array vendors usually provide comprehensive integrated CAD tools for logic simulation, testability analysis and autolayout and thus decrease the probability of error.

Although gate array technology is not always the best in each comparison presented above, it is the best approach for the NuFTbus BIU design when considering all the comparisons. Actually, in today's semi-custom IC market, gate arrays lead cell-based systems in the market share by 4 to 1, though the difference between them diminishes as IC technologies evolve [17].

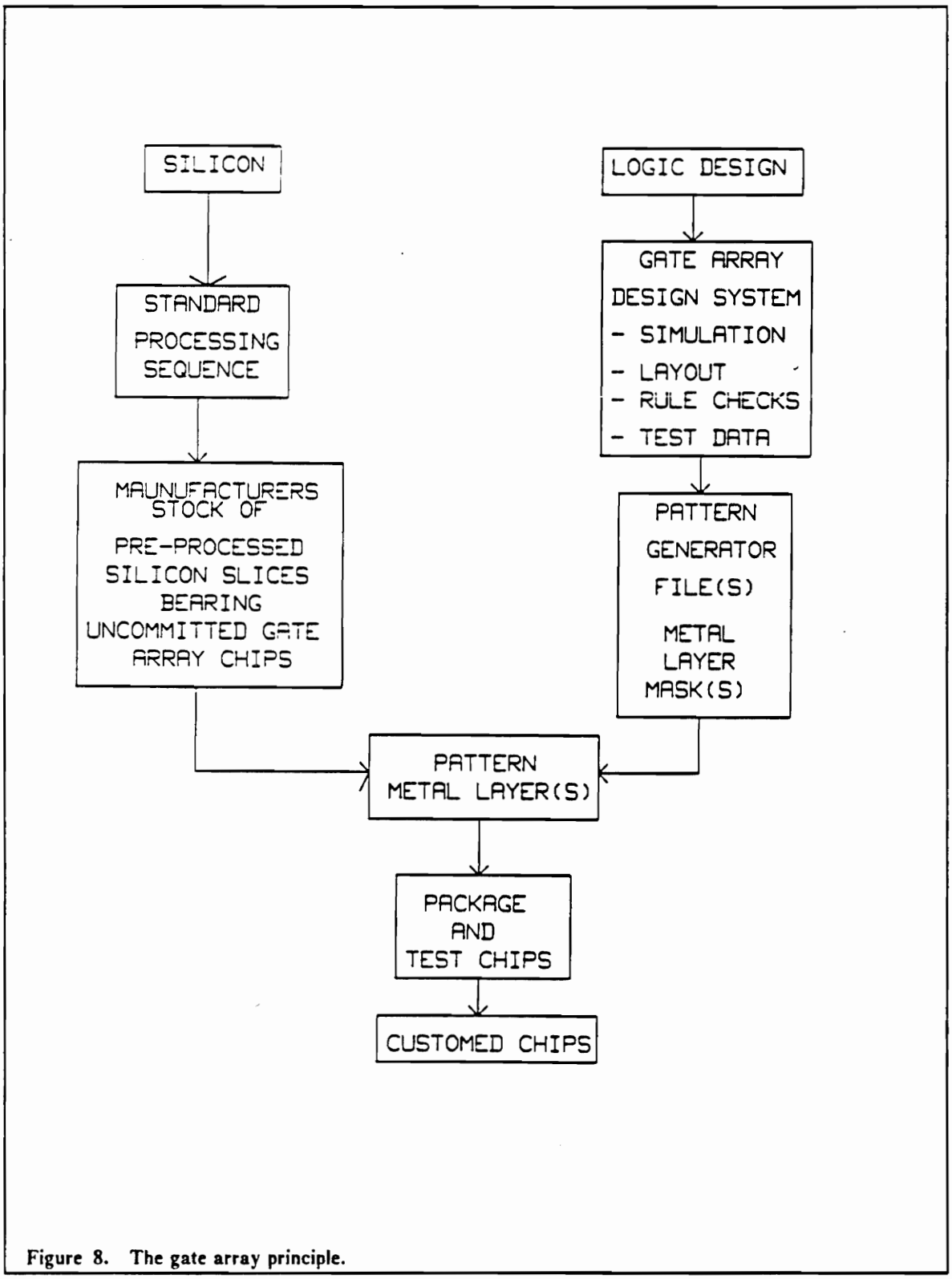


Figure 8. The gate array principle.

5. Introduction to CAD Tools Used

This chapter introduces the CAD tools used during the NuFTbus BIU development process. First, the gate array IC design system HIGHLAND from UTMC (United Technology Microelectronics Center) is introduced, and then CAD tool PEG used for designing finite state machines are discussed. This is followed by a discussion of LOGEN which is used to generate the logic in McLDL (Microelectronics Center Logic Description Language) syntax. Examples for each are provided.

5.1 Highland Design System

The HIGHLAND design system is United Technologies Microelectronics Center's (UTMC) proprietary CAD tool for Gate Array IC design. This VAX/VMS-based HIGHLAND system contains a common database and an integrated set of tools to perform netlist capture, simulation, automatic layout, and testability analysis.

5.1.1 Design Flow: The high level steps in chip implementation using the HIGHLAND Design System to design gate array ICs are listed below and shown schematically in Figure 9 on page 45 [18].

1. Initial specification for the design is written.
2. Chip is designed using the HIGHLAND Design System.
3. An ARCHIVE of the resultant design is sent to UTMC.
4. Design information is translated into formats required for mask generation and automatic tester programs.
5. Masks are built.
6. Parts are fabricated.
7. Wafers are tested prior to assembly.
8. Wafers are cut into chips and put in packages.
9. Packaged parts are tested.
10. Good parts are shipped to the customer.

Most of the activities that involve circuit designs in gate array IC design are in step 1 and step 2. In step 2, the circuit designers follow the HIGHLAND design flow shown in Figure 10 on page 46. This design procedure is also listed below.

1. **HIGHLAND**: Typing HIGHLAND to invoke the HIGHLAND Design System.
2. **TRANS**: Translating the .LDL file (netlist) into .CMF (Chip Module File) for simulation.
3. **SIL**: Compiling the .SIL file (Simulation Input Language) into .SCC (Simulation Command Control) as input stimuli for the simulator.
4. **SIM**: Exercising the circuit design with expected inputs to verify design correctness.
5. **FSIM/XC**: Fault-grading the circuit design to show what nodes cannot have stuck-at faults detected based on user input stimuli.
6. **LAYOUT**: Performing the autolayout (placement and routing) program.
7. **SIM**: Postlayout logic simulation.
8. **FSIM/XC**: Postlayout fault-grading simulation.
9. **PAVE**: Pre-Archive-Verification.
10. **ARCHIVE**: Setting aside on tape the HIGHLAND files and information necessary to document, fabricate, and test an integrated circuit.

The .LDL file written in the McLDL logic description language and the .SIL file written in a simulation input language are further explained in the following sections.

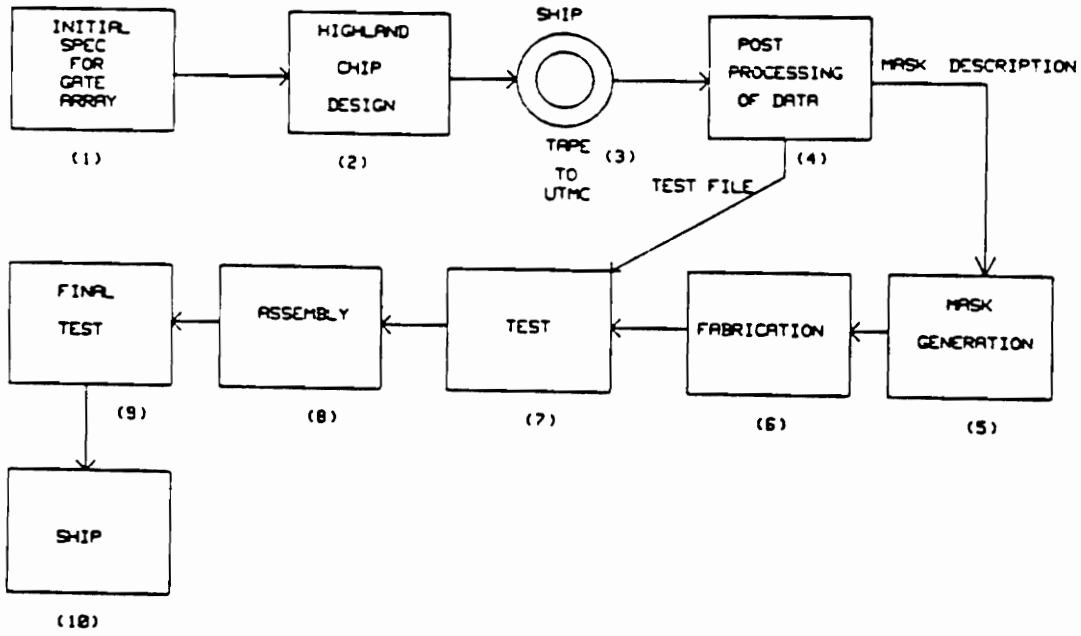


Figure 9. Steps in implementing gate array ICs.

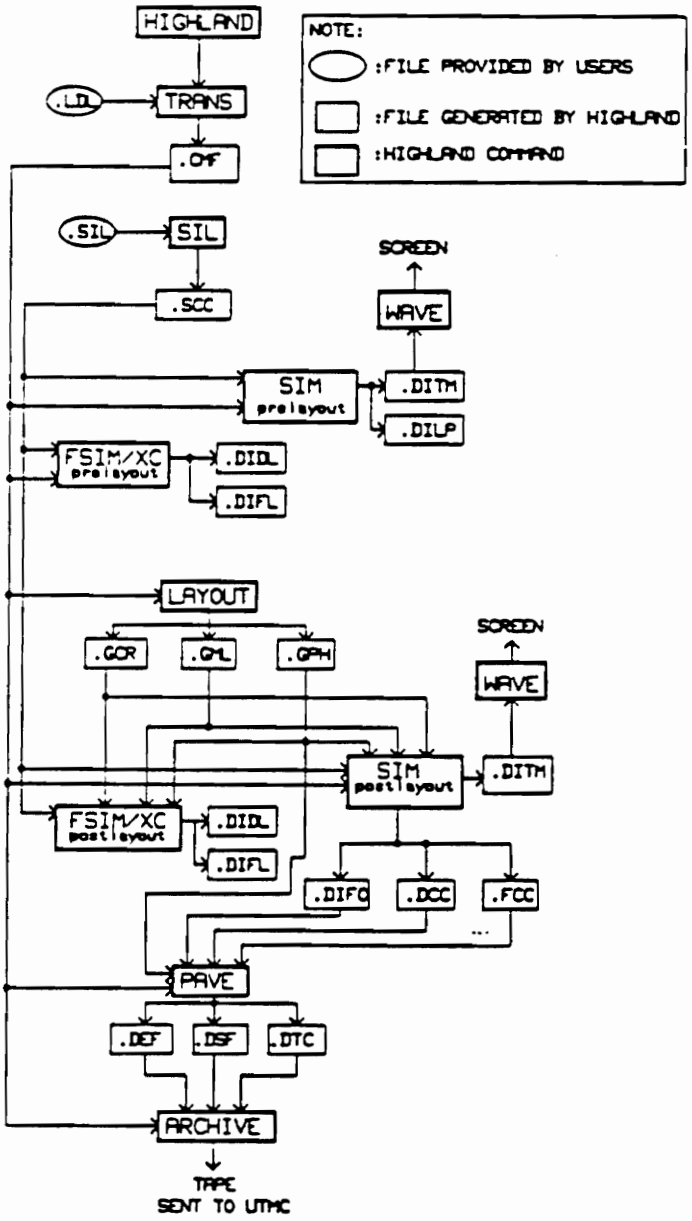


Figure 10. HIGHLAND design flow.

5.1.2 McLDL Logic Description Language: The McLDL Logic Description Language (McLDL) is a straightforward textual network description language [18]. Circuits are designed by entering netlists described by McLDL with a text editor, or by running LOGEN (logic generator). The resultant files are named with a .LDL extension. Included in the .LDL file is a module name, input signal list, output signal list, bidirectional signal list, followed by the network description. The actual header format for a single module is given in Figure 11 on page 48.

The example shown in Figure 12 on page 49 is a McLDL netlist, and its corresponding circuit configuration shown in Figure 13 on page 50 demonstrates some aspects of the logic description language. The lines following the "*" are treated as comments. The word EXAMPLE after the keyword MODULE is the name of the chip or the submodule of the chip. The INPUT and OUTPUT keywords indicate the direction of the signal with respect to the module. The BIDIRECT is not present since there is no bidirectional signals in this example. The keyword NETWORK indicates that the network description follows and ENDMODULE indicates the completion of the module description. Each entry in the network description is comprised of a unique occurrence name, such as U1, U2, NOPE and YEP, a part name, such as NAND3, FOO and INV4, input signal list, such as IN1, FRED, OUTBIT and NOPE, output signal list, such as FRED and OUTBIT, or bidirectional signal list. To distinguish between the end of one signal list and start of another, a ":" is used as a delimiter. If an output signal from a part is not needed for a specific application, it must be explicitly declared as a non-connection. This is done by placing an @ character in place of the appropriate signal. If a part has only one output, the output signal is optional and the unique occurrence will be treated as the output signal for that part provided that the output signal is not present, such as NOPE and YEP [18].

5.1.3 SIL Simulation Input Language: UTMC's simulators all run with a single common simulation input language SIL. SIL is a very high level language which is capable of not only simple change card input stimuli but can also have conditional and behavioral constructs. Change card input stimuli is a construct used to cause a signal to change to a specified state at a specified instance of

```
MODULE TEST
INPUT IN1 IN2 ...
OUTPUT OUT1 OUT2 ...
BIDIRECT BI1 BI2 ...
NETWORK
network description
  .
  .
  .
ENDMODULE
```

Figure 11. Header format of a McLDL program.

```
* McLDL syntax example
MODULE EXAMPLE
INPUT IN1 IN2 ARG
OUTPUT OUTBIT NOPE YEP
NETWORK
U1    NAND3  IN1 IN2 ARG : FRED
U2    FOO    FRED ARG : @ OUTBIT
NOPE  INV4   OUTBIT
YEP   INV4   NOPE
* This is a comment
ENDMODULE
```

Figure 12. An example of a McLDL circuit description.

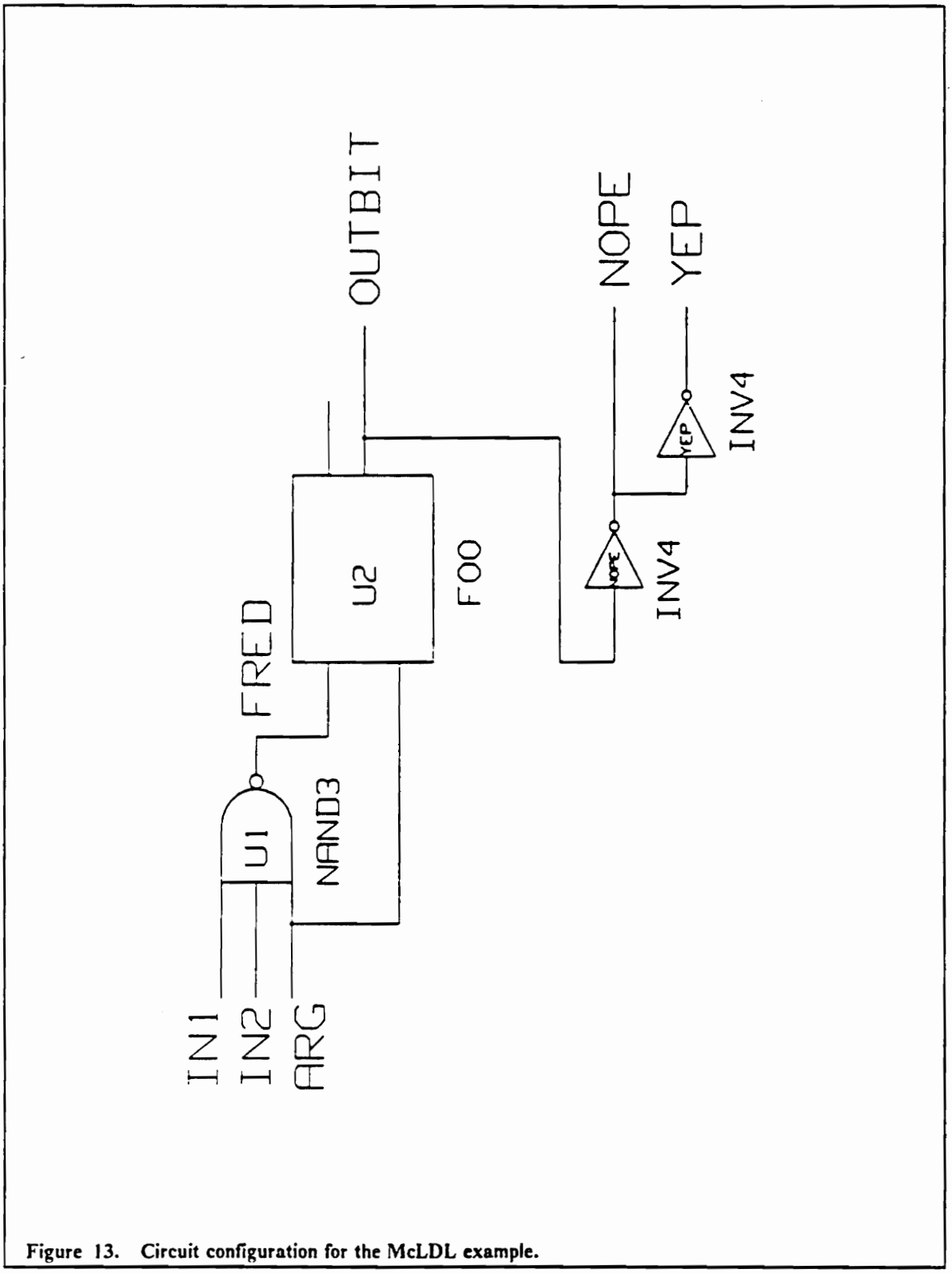


Figure 13. Circuit configuration for the McLDL example.

time. The following features are also available in SIL: IF statements, FOR loops, RETURN statements, a full complement of Boolean and arithmetic operators, a COUNT command, user definable blocks of SIL codes, and extensive variable manipulation. Details of syntax for SIL are in the HIGHLAND Reference Manual [18].

It is possible to take the features of SIL itself and create a **behavioral** model of the actual gate level module being simulated. For example, it is possible to use SIL to write behavioral descriptions of circuits (the M[i,j,k] defined in Figure 14 on page 53), then write a SIL block (the sim in Figure 14 on page 53) to generate signals that use inputs (i, j, k) to the behavioral description and stimuli (A, B, C) for the McLDL description (module M in Figure 15 on page 54) of the circuit being simulated. It is then possible to verify the McLDL (module M) using an XOR (in module COMPARE) to compare the unit delay (by specifying Delay Condition A when invoking the simulator) simulation output (X) with the values (Y) the SIL description generates. If the output of the XOR is 0, then the McLDL description is consistent with the SIL behavioral description. That is, the gate level implementation of the circuit (module M) is consistent with the behavioral model of the circuit. This saves time when many simulation results must be verified. For instance, there will be more than 500,000 lines of output if every possible combination were simulated over an 8-function, 8-bit ALU.

The SIL syntax in Figure 14 on page 53 is explained as follows. A GLOBAL block is where global definitions of SIL variables are made. GLOBAL variables are visible to DEFINE and EXPAND blocks. The keyword STEP is used to specify the time step for the simulation. In this example, the time step is 100 ns; that is the input stimuli is changed every 100 ns. The variable d is used to specify time delay in the behavioral model M[i,j,k]. The keyword EXPAND indicates the beginning of the evaluation of the SIL code. **CHANGE A : i : T** means change signal A to state i at time T. The terminate last command stops the output of signal changes at the last time a signal change. The save and display commands indicate to the simulator what signals to save or display. A DEFINE block is a set of code the designer may call from other DEFINE blocks, the GLOBAL block, or the EXPAND block. DEFINE blocks have the same syntax as the EXPAND and the

GLOBAL block except that an optional identifier list is used for parameter passing (in this example, [i,j,k]).

5.2 PEG

PEG (PLA Equation Generator) is a design tool used to produce designs of finite state machines (FSM). It compiles high level language descriptions of finite state machines into the logic equations needed to implement a design. For this thesis, PEG is used to design the FSM controllers (BTC and NBC) in the BIU. Logic equations generated by PEG describe only the combinational logic in the FSM. In order to complete the FSM design, the users have to add synchronous logic such as D-type flip-flops to latch input and output signals. This will be shown in the example in section 5.3.

PEG programs are isomorphic to Moore machine state diagrams. There is a one-to-one correspondence between states in a state diagram and state definitions in the corresponding PEG program. The translation from state diagrams to PEG programs is simple and straightforward.

Designing with PEG provides a number of advantages over the traditional pencil-and-paper approach to FSM design. PEG's high level language enables designs and design changes to be quickly implemented. PEG programs provide easy-to-understand documentation with clear control flow. PEG does the tedious and error-prone bookkeeping task of generating output and next state bits as a function of current state bits. It checks for design errors and eliminates redundant terms in logic equations. Details of PEG's grammar are in the Berkeley CAD Tools User's Manual [19].

5.2.1 An Example Using the PEG Language: Figure 16 on page 56 shows an example of a simple finite state machine used to decode the inputs a, b and c into 0, 1, 2, 3 or "other". The corresponding state diagram is shown in Figure 17 on page 57.

```
/* This SIL generates simulation stimuli and
   calls the SIL behavioral description */

GLOBAL declaration;
STEP=100;
d=1;
END declaration;

EXPAND sim;
T=1;
FOR i = 0 TO 1 DO
    FOR j = 0 TO 1 DO
        FOR k = 0 TO 1 DO
            CHANGE A : i : T;
            CHANGE B : j : T;
            CHANGE C : k : T;
            CHANGE Y : M[i,j,k] : T+d;
            T = T + STEP;
        END;
    END;
END;

terminate last;
save all;
display all;
END sim;

/* This is the SIL behavioral description
of the McLDL module M  and is call by sim */

DEFINE M[i,j,k];
s = i AND j;
RETURN s OR k;
END;
```

Figure 14. Example of SIL program.

```
/* The module to compare SIL
output with simulation output.
Y is the output of the
SIL behavioral description.
X is the output of the circuit to
verify.
Z must be 0 if the SIL and
McLDL are consistent */

MODULE COMPARE
INPUT A B C Y
OUTPUT Z
NETWORK
  U1 M  A B C : X
  U2 XOR X Y : Z
ENDMODULE


/* This is the module to verify and
is the submodule used in module compare */

MODULE M
INPUT A B C
OUTPUT X
NETWORK
  U1 AND2 A B : S
  U2 OR2 C S : X
ENDMODULE
```

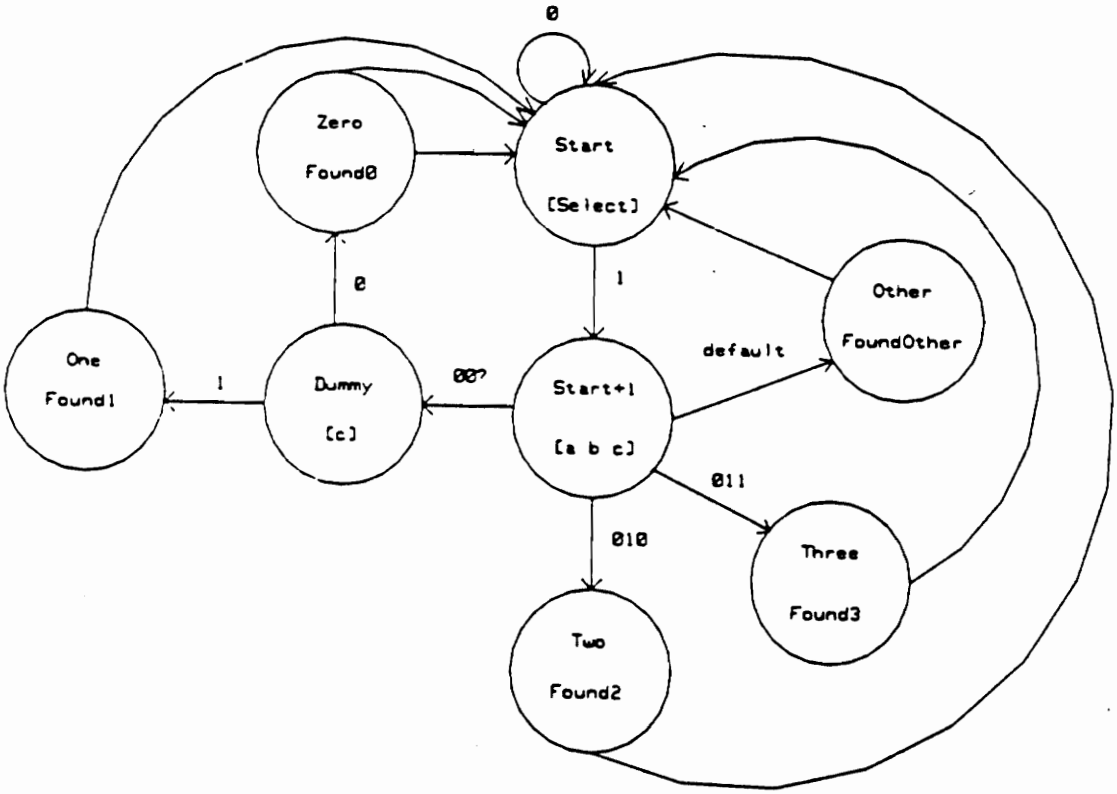
Figure 15. McLDL modules to compare and verify.

The lines follow the "--" are treated as comments. Since this finite state machine has five inputs, they are declared in the INPUTS statement. Similarly, the outputs are declared in the OUTPUTS statement. The first state in the example is labeled with the identifier **Start**. This label is necessary only because of the GOTO statements in other states back to state 0. Flow of control in PEG program is sequential unless otherwise specified by the IF-THEN-ELSE control construct or by the CASE statement. The IF-THEN-ELSE control construct is used to provide two-way branches based only on a single input signal while branches based on more than one input signal are handled by the CASE statement. Since no control information other than loop is present for state Start (state 0), the program steps sequentially to state Start + 1 (state 1) when Select is high. The "?" in the arguments of the CASE statement stands for don't care condition. In this example it can be replaced by the two arguments: 0 0 0 and 0 0 1 and is used here only for demonstration. By default, all of the output signals will be in low state unless they are asserted (by the ASSERT statement) in a certain state such as in Found0 in state Zero (state 3). If a signal must be high across several states, then it must be asserted high at each state where it is intended to be high. RESET in this program indicates that when the RESET signal is asserted, the state machine jumps to the top of the program, which in this case is named Start.

When invoking the PEG compiler, including the option -t, causes truth table for the finite state machine to be generated and put into the file PEG.SUMMARY. This file for the FSM in Figure 16 on page 56 is shown in Figure 18 on page 59. The mapping from truth table column labels to actual signal names is given in the lists of input and output signals which precede the truth table. The i columns stand for the input vectors (corresponding to different states) composed of the RESET, Select, a, b, and c input signals. Similarly, the o columns stand for the output vectors composed of the Found0, Found1, Found2, Found3 and FoundOther output signals. The s columns and n columns represent the present state vectors and next state vectors, respectively. The dashes present in the input vectors mean **don't care** while those in the output vectors mean the corresponding output signals are not asserted in those states and should be 0s by default. To the right of the truth table are the names of the states described by the rows of the table.

```
-- This is an example of FSM for decoding
-- inputs a,b, and c into
-- 0,1,2,3, or "other"

INPUTS    :  RESET Select a b c;
OUTPUTS   :  Found0 Found1 Found2 +
             Found3 FoundOther;

Start     :  -- This is the reset state - state 0
             IF NOT Select THEN LOOP;

          :  CASE(a b c) --Second state - state 1
               0 0 ? => Dummy; --A don't care
               0 1 0 => Two;
               0 1 1 => Three;
             ENDCASE=> Other;

Dummy     :  IF c THEN One; -- state 2

Zero      :  ASSERT Found0; GOTO Start; -- state 3

One       :  ASSERT Found1; GOTO Start; -- state 4

Two       :  ASSERT Found2; GOTO Start; -- state 5

Three     :  ASSERT Found3; GOTO Start; -- state 6

Other     :  ASSERT FoundOther; GOTO Start; -- state 7
```

Figure 16. Example of a PEG program describing a FSM.



NOTE:

1. At states Start, Start+1, and Dummy, no output is asseted.
2. At states Zero, One, Two, Three and Other, the corresponding outputs Found0, Found1, Found2, Found3 and FoundOther are asserted accordingly.

Figure 17. State diagram for the PEG example.

This PEG.SUMMARY file generated by PEG is fed into the format translator program called TSF which then produces the input file required by LOGEN. TSF will be documented in section 5.3.4. By doing so, LOGEN can generate the combinational logic in the finite state machine in McLDL syntax (.LDL file).

5.3 LOGEN

LOGEN is a logic generation system. It minimizes logic, synthesizes random combinational logic, and generates combinational logic for state machines in McLDL syntax (.LDL file) [20].

LOGEN's data source is a user-provided logic description based on a logic table [20]. The logic description consists of a series of input vectors, each followed by an output vector. Each input vector can actually represent a set of vector patterns, since don't cares (x) are allowed. Each output vector specifies the required output when the given input occurs. Examples for combinational machine and finite state machine will be presented and explained in the sections that follow.

5.3.1 The Input Syntax: The user describes the desired logic by providing a table of inputs and outputs. All of the like inputs and outputs are in the same column. Each row contains an input vector followed by an output vector. Don't cares are allowed in either. LOGEN will build a combinational machine such that whenever the given input vectors are encountered the desired output vector will result.

Command lines starting with a period "." are used to indicate the number of inputs and outputs, and where a module ends. Each ofthese commands is required, except the end command can be left off if there is only one module in the file.

Command lines starting with the pound sign "#" are used for both comments and other optional information. Pound commands are used for naming the module, naming the input and output bits,

```
SUMMARY INFORMATION GENERATED BY
PEG FROM FILE fsm.peg
INPUTS:
   i00:  RESET
   i01:  Select
   i02:  a
   i03:  b
   i04:  c
   s00:  InSt0* (msb)
   s01:  InSt1*
   s02:  InSt2* (lsb)
OUTPUTS:
   n02:  OutSt2* (lsb)
   n01:  OutSt1*
   n00:  OutSt0* (msb)
   o00:  Found0
   o01:  Found1
   o02:  Found2
   o03:  Found3
   o04:  FoundOther
State Table
     i i i i i  s s s  n n n  o o o o o
     0 1 2 3 4  0 1 2  2 1 0  0 1 2 3 4
    ------------------------------------
     1 - - - -  0 0 0  0 0 0  - - - - -  Start
     0 0 - - -  0 0 0  0 0 0  - - - - -  Start
     0 1 - - -  0 0 0  1 0 0  - - - - -  Start
     1 - - - -  0 0 1  0 0 0  - - - - -  Start+1
     0 - 0 0 -  0 0 1  0 1 0  - - - - -  Start+1
     0 - 1 - -  0 0 1  1 1 1  - - - - -  Start+1
     0 - 0 1 0  0 0 1  1 0 1  - - - - -  Start+1
     0 - 0 1 1  0 0 1  0 1 1  - - - - -  Start+1
     1 - - - -  0 1 0  0 0 0  - - - - -  Dummy
     0 - - - 0  0 1 0  1 1 0  - - - - -  Dummy
     0 - - - 1  0 1 0  0 0 1  - - - - -  Dummy
     1 - - - -  0 1 1  0 0 0  1 - - - -  Zero
     0 - - - -  0 1 1  0 0 0  1 - - - -  Zero
     1 - - - -  1 0 0  0 0 0  - 1 - - -  One
     0 - - - -  1 0 0  0 0 0  - 1 - - -  One
     1 - - - -  1 0 1  0 0 0  - - 1 - -  Two
     0 - - - -  1 0 1  0 0 0  - - 1 - -  Two
     1 - - - -  1 1 0  0 0 0  - - - 1 -  Three
     0 - - - -  1 1 0  0 0 0  - - - 1 -  Three
     1 - - - -  1 1 1  0 0 0  - - - - 1  Other
     0 - - - -  1 1 1  0 0 0  - - - - 1  Other
```

Figure 18. PEG.SUMMARY file for the FSM.

and providing the number of state bits (if a FSM is to be built). The "#" sign should be followed by at least one space if the line is a comment line.

The characters 1 and 0 are used to describe these states, and "-", "_", "x", or "X" can be used for don't cares. Any white space (blanks), except when required as a delimiter, is ignored. It is assumed that all commands will appear on a new line. In general, this syntax was chosen to be compatible with ESPRESSO (a logic table minimization tool) [18]. The example in Figure 19 on page 61 shows some aspects of the LOGEN syntax for a combinational machine. The example for a finite state machine together with its .LDL file will be discussed in section 5.3.3.

5.3.2 Consistency Check: Because don't cares are allowed when providing the input vectors to LOGEN, it is possible that the user may unknowingly specify two input vectors which overlap and have different output vectors. Foe example:

< input vectors >	< output vectors >
0 0 0 1 0 - - - 0 1 - 1	0 1 1 0 1
- 0 0 1 - 1 1 1 - - 1 1	0 1 1 1 0

The 000101110111 is an overlap of the above input vectors, yet two of the output bits in their output vectors are different. This inconsistency is checked for by LOGEN and will be flagged with an error message (equivalent input vectors with different output vectors).

The normal procedure for generating the hardware for the state machine would next require the user to invoke EQNTOTT (an equation to turth table conversion CAD tool) to generate a truth table corresponding to the combinational logic of the FSM. The truth table would be used as input to a PLA generating program like MPLA. Failing the consistency checking is the why reason we do not pipe the logic equations into EQNTOTT to generate the input file needed by LOGEN. The truth table produced by EQNTOTT cannot pass this consistency checking, and consequently forces a switch to the use of the information in the file PEG.SUMMARY. Rather than the use of

```
#MODULE adder

# This module is a 2-bit by 2-bit binary adder
# (with no carry input)

#iv [1:4] input11 input12 input21 input22
#ov [1,2] out1 out2
#ov [3]   out3

.input 4
.output 3

     0000      000
     0001      001
     0010      010
     0011      011
     0100      001
     0101      010
     0110      011
     0111      100
     1000      010
     1001      011
     1010      100
     1011      101
     1100      110
     1101      100
     1110      101
     1111      110

.end
```

Figure 19. Example of the LOGEN input syntax.

the EQNTOTT and ESPRESSO as proposed by Paranjape, TSF is used to provide input which LOGEN can accept as input. TSF will be further documented in section 5.3.3.

5.3.3 State Machine Generation: When designing a finite state machine, the present state vectors in the logic table used by LOGEN must occur first in the input vectors, and the next state vectors must occur first in the output vectors as shown below:

< present state > < random inputs > < next state > < random outputs >

Furthermore, each module, using the #state command, must declare how many bits are in the state vector. Then the parser in LOGEN can process this module as a finite state machine.

When LOGEN produces the .LDL file for the finite state machine, it generates only the combinational logic. To convert this to a full state machine, the following steps must be taken:

1. Add to the LDL the same number of flip-flops as there are bits in the state vector.

2. Remove the present state input bits from the input list of the LDL module. Connect these signals to the Q outputs of the flip-flops added in step one.

3. Remove the next state output bits from the output list of the LDL module. Connect these signals to the D inputs of the respective flip-flops.

4. Connect the clocks of the flip-flops together and put the clock signal in the input list to the module.

These steps used to generate a complete finite state machine are further described in the next section. This conversion can be done manually or by the program called FIX which was written to

fix these and other problems in LOGEN. When the second method is used, it adds in the DFFAC flip-flops supplied by the cell library. FIX will be documented in more details in section 5.3.5.

5.3.4 Connection Between PEG and LOGEN: LOGEN requires a certain format of input. The syntax has been briefly introduced in section 5.3.1. Since the truth table generated by PEG is in-compatible with the format required by LOGEN, a program called TSF running under the UNIX operating system was written by the author to transform the format from a PEG.SUMMARY file to the format which can be accepted by LOGEN, and thereby make the connection between PEG and LOGEN.

Figure 20 on page 64 shows an input file for LOGEN, which was transformed from the PEG.SUMMARY in Figure 18 on page 59 by the program TSF. There are three bits in the state vector, which the module declares it using the #state 3 command.

Figure 21 on page 65 is the .LDL file generated by LOGEN using the file in Figure 20 on page 64 as input. This is only the combinational logic of the finite state machine. Figure 22 on page 66 shows the complete finite state machine made from the file in Figure 21 on page 65 by taking those steps described in previous section. And the gate level circuits are presented in Figure 23 on page 67.

5.3.5 Problems in LOGEN and Solutions: LOGEN can generate combinational logic from an input file of a specified format. Although its output file is claimed to be of the McLDL file format, there are problems that cause errors when the file is being translated by HIGHLAND. First, some cells in the McLDL file presently do not exist in the cell library. These include cells such as BUF24 (a buffer with two complemented outputs with different driving capabilities), NOR4 and NAND34 (a 34-input NAND gate), etc.. Secondly, some cells requested in the LOGEN output exceed their driving capabilities. For example, the output of an INV1 cell may be connected to more than eight inputs of other gates by the LOGEN output, which is beyond the INV1's driving capability.

```
#MODULE fsm
#iv [4] RESET
#iv [5] Select
#iv [6] a
#iv [7] b
#iv [8] c
#ov [4] Found0
#ov [5] Found1
#ov [6] Found2
#ov [7] Found3
#ov [8] FoundOther
.input 8
.output 8
#state 3
# Truth Table derived from file PEG.SUMMARY
#    s s s  i i i i i  n n n  o o o o o
#    0 1 2  0 1 2 3 4  0 1 2  0 1 2 3 4
#  ---------------------------------------
     0 0 0  1 - - - -  0 0 0  0 0 0 0 0
     0 0 0  0 0 - - -  0 0 0  0 0 0 0 0
     0 0 0  0 1 - - -  0 0 1  0 0 0 0 0
     0 0 1  1 - - - -  0 0 0  0 0 0 0 0
     0 0 1  0 - 0 0 -  0 1 0  0 0 0 0 0
     0 0 1  0 - 1 - -  1 1 1  0 0 0 0 0
     0 0 1  0 - 0 1 0  1 0 1  0 0 0 0 0
     0 0 1  0 - 0 1 1  1 1 0  0 0 0 0 0
     0 1 0  1 - - - -  0 0 0  0 0 0 0 0
     0 1 0  0 - - - 0  0 1 1  0 0 0 0 0
     0 1 0  0 - - - 1  1 0 0  0 0 0 0 0
     0 1 1  1 - - - -  0 0 0  1 0 0 0 0
     0 1 1  0 - - - -  0 0 0  1 0 0 0 0
     1 0 0  1 - - - -  0 0 0  0 1 0 0 0
     1 0 0  0 - - - -  0 0 0  0 1 0 0 0
     1 0 1  1 - - - -  0 0 0  0 0 1 0 0
     1 0 1  0 - - - -  0 0 0  0 0 1 0 0
     1 1 0  1 - - - -  0 0 0  0 0 0 1 0
     1 1 0  0 - - - -  0 0 0  0 0 0 1 0
     1 1 1  1 - - - -  0 0 0  0 0 0 0 1
     1 1 1  0 - - - -  0 0 0  0 0 0 0 1
.end
```

Figure 20. Input file to LOGEN derived from file PEG.SUMMARY.

```
MODULE fsm
INPUT    IN1 IN2 IN3 RESET Select a b c
OUTPUT   OUT1 OUT2 OUT3 Found0 Found1 +
         Found2 Found3 FoundOther
NETWORK
* input buffering *
I1 BUF11 IN1 : BIN2 BIN1
I2 INV1 IN2 : BIN3
I3 BUF11 IN3 : BIN5 BIN4
I4 INV1 RESET : BIN6
I5 INV1 a : BIN7
I6 INV1 c : BIN8
* sub min term generation *
I7 NOR3 BIN1 IN2 RESET : SN1
I8 NOR2 BIN1 BIN3 : SN2
* output bit 1 *
I9 NAND3 a BIN4 SN1 : N1
I10 NAND3 b BIN4 SN1 : N2
I11 NAND4 c BIN6 BIN5 SN2 : N3
I12 NAND3 N3 N2 N1 : OUT1
* output bit 2 *
I13 NAND2 BIN5 BIN3 : N4
I14 NAND5 BIN8 b BIN7 BIN4 SN1 : N5
I15 NAND2 BIN4 SN2 : N6
I16 NAND6 BIN2 N6 N3 N5 N4 BIN6 : N7
I17 INV1 N7 : OUT2
* output bit 3 *
I18 NAND3 Select BIN5 SN1 : N8
I19 NAND4 BIN8 BIN6 BIN5 SN2 : N9
I20 NAND4 N9 N5 N1 N8 : OUT3
* output bit 4 *
I21 INV1 N6 : Found0
* output bit 5 *
I22 NAND3 BIN5 BIN3 BIN1 : N10
I23 INV1 N10 : Found1
* output bit 6 *
I24 NAND3 BIN4 BIN3 BIN1 : N11
I25 INV1 N11 : Found2
* output bit 7 *
I26 NAND3 BIN5 IN2 BIN1 : N12
I27 INV1 N12 : Found3
* output bit 8 *
I28 NAND3 BIN4 IN2 BIN1 : N13
I29 INV1 N13 : FoundOther
ENDMODULE
```

Figure 21. Combinational logic in the FSM generated by LOGEN.

```
MODULE fsm
INPUT   RESET CLK Select a b c
OUTPUT  Found0 Found1 Found2 Found3 FoundOther
NETWORK
* flip-flops for finite state machine
U<1..3> DFF CLOCK OUT<1..3> : IN<1..3> @
U4 BUF11 CLK : @ CLOCK
* input buffering *
I1 BUF11 IN1 : BIN2 BIN1
I2 INV1 IN2 : BIN3
I3 BUF11 IN3 : BIN5 BIN4
I4 INV1 RESET : BIN6
I5 INV1 a : BIN7
I6 INV1 c : BIN8
* sub min term generation *
I7 NOR3 BIN1 IN2 RESET : SN1
I8 NOR2 BIN1 BIN3 : SN2
* output bit 1 *
I9 NAND3 a BIN4 SN1 : N1
I10 NAND3 b BIN4 SN1 : N2
I11 NAND4 c BIN6 BIN5 SN2 : N3
I12 NAND3 N3 N2 N1 : OUT1
* output bit 2 *
I13 NAND2 BIN5 BIN3 : N4
I14 NAND5 BIN8 b BIN7 BIN4 SN1 : N5
I15 NAND2 BIN4 SN2 : N6
I16 NAND6 BIN2 N6 N3 N5 N4 BIN6 : N7
I17 INV1 N7 : OUT2
* output bit 3 *
I18 NAND3 Select BIN5 SN1 : N8
I19 NAND4 BIN8 BIN6 BIN5 SN2 : N9
I20 NAND4 N9 N5 N1 N8 : OUT3
* output bit 4 *
I21 INV1 N6 : Found0
* output bit 5 *
I22 NAND3 BIN5 BIN3 BIN1 : N10
I23 INV1 N10 : Found1
* output bit 6 *
I24 NAND3 BIN4 BIN3 BIN1 : N11
I25 INV1 N11 : Found2
* output bit 7 *
I26 NAND3 BIN5 IN2 BIN1 : N12
I27 INV1 N12 : Found3
* output bit 8 *
I28 NAND3 BIN4 IN2 BIN1 : N13
I29 INV1 N13 : FoundOther
ENDMODULE
```

Figure 22. The complete finite state machine.

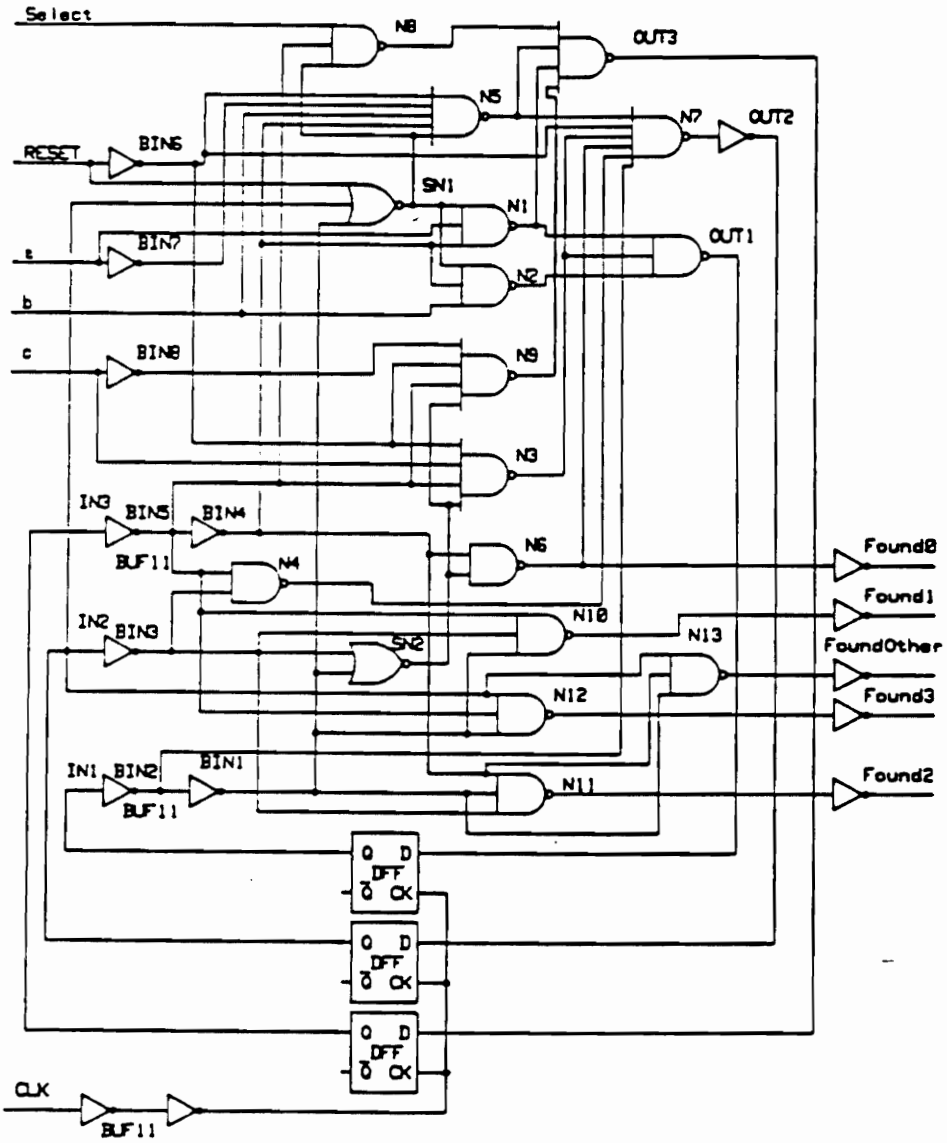


Figure 23. Gate level circuits for the FSM.

This problem can be fixed manually using any text editor. However, this mechanical procedure is error-prone and very difficult to debug. So, a preprocessor program has been written in VAX-11 C (VMS) to fix most of the problems. The driving capability problems in cells INV4, BUF4? and BUF?4 still have to be solved manually because there is no cell which has greater drive capability and can be used to replace them. The function of this program can be divided into two categories. The first one is to fix the first problem stated in the last paragraph. The function is performed by invoking the preprocessor FIX with the following command as:

```
$ FIX infile.LDL outfile.LDL -dmg
```

The switch **-dmg** is used to specify the first function that the FIX program is to perform. That is, the replacement of nonexistent cells with existing library cells using DeMorgan's Law. The second function performed is to solve the driving capability problem. By invoking the program with the switch **-drv**, the program will ask the user to type in the signal name which exceeds its driving capability. Cells are fixed one by one until the user types **quit**.

For example, if the following is part of a .LDL file and SN2 is the signal that exceeds the driving capability of the NOR2 gate and I23 is the cell that doesn't exist in the cell library, then FIX is used to fix these problems.

```
I6 NOR2 BIN2 BIN3 : SN2
I23 NAND9 BIN1 BIN2 BIN3 BIN4 SN1 SN2 SN3 SN4 N1 : N5
```

After running FIX, they will look like the following,

```
* I6 NOR2 BIN2 BIN3 : SN2
I61 NOR2 BIN2 BIN3 : TP1
I62 BUF11 TP1 : @ SN2
* I23 NAND9 BIN1 BIN2 BIN3 BIN4 SN1 SN2 SN3 SN4 N1 : N5
```

I231 NAND3 BIN1 BIN2 BIN3

I232 NAND6 BIN4 SN1 SN2 SN3 SN4 N1

I233 OR2 I231 I232 : N5

5.4 Static Hazards

Because of the different gate delays and different signal propagation paths through the state machine, static and dynamic hazards are present in the state machine. Moreover, some of these signal are used to set and reset SR flip-flops, and therefore make those flip-flops store the wrong states.

Although methods are presented in logic design text books to eliminate static hazards in simple circuits, they do not help in this BIU design since this real circuit is quite complicated and is automatically generated by the CAD tool LOGEN, so that we are not able to do anything about how the algorithm assigns states. Therefore, all output signals of state machines are latched into flip-flops, the clock for the flip-flop is the clock inverted from that which controls the finite state machines. This is done automatically when executing the **FIX** program described in the previous section. Since hazards occur right after state transitions and will last for at most the longest propagation delay time in the state machine, the clock on the extra flip-flop latches in the output signals of the state machine at the time long after the state transition and therefore avoids the hazards.

6. Circuit Design and Simulation Results

In Chapter 3, an overview and basic functions of the BIU was presented. In this chapter, each circuit module in the BIU will be explained in more detail by showing its circuit diagram and simulation results.

Since the circuitry in the NuFTbus interface unit is too big to be put on any single gate array presently offered by UTMC, there are two chips in the design. One is called the A-chip and the other is the B-chip. The circuits are designed by entering on the HIGHLAND Design System, netlists written in the McLDL logic description language (McLDL is explained in Section 5.1.2). Every circuit module is compiled and simulated on the HIGHLAND Design System using UTMC's UTD-R logic cells. These cells are based on a 1.5 micron CMOS radiation-hardened technology. All the cells shown in the schematics are from this library with their names on them if there are special cells, such as an 8-bit comparator (UT682) and a 4-bit up-down counter (UT169). Since the simulation was run before layout (i.e., before routing among modules), the delay is only that of the propagation delay in the N and P transistors and the interconnect delay between the transistors. Furthermore, the delay condition was specified as CONDITION D, i.e., the low-conductance (SLOW) case which correlates to high-temperature, low-voltage conditions (+ 125 degrees C and 4.5 volts) and fan-out loading for the military range [23]. This gives a worst case result but ignores for the present the delay caused by intramodule wiring. Intramodule wiring

delays can be accounted for once the layout is performed, thus necessitating a second pass design phase need to further verify correct operation. The following sections describe the circuit functions and the simulation results.

6.1 A-Chip

6.1.1 Block Diagram of the A-Chip

As shown in Figure 24 on page 72, the A-chip consists of the following circuits: the arbitration module, the board transfer controller (BTC), the NuFTbus controller (NBC), the control path, the bus status detection logic, the detection of bus locking, and the EDAC sharing logic. Each module is explained in the following sections.

6.1.2 Arbitration Module

The NuFTbus arbitration module can tolerate any one-bit error and arbitrate as if there were no stuck-at faults on the ID lines [14]. The arbitration mechanism is based on a priority arbitration scheme (the module with the lowest ID code wins bus mastership). However, priority on the NuFTbus is different from those of buses that have strict priority arbitration (such as VMEbus). The NuFTbus arbitration scheme distributes the bus bandwidth evenly among all modules and is thus referred as a fair priority arbitration.

When a module needs to start a transaction, it first competes for bus mastership by asserting (pulling low) the open drain RQST line. However, a module may only assert RQST when it sees this line that is in the deasserted state (high); this prevents a higher priority module from preempting those modules already in the arbitration contest. Whenever RQST is asserted by a module, arbi-

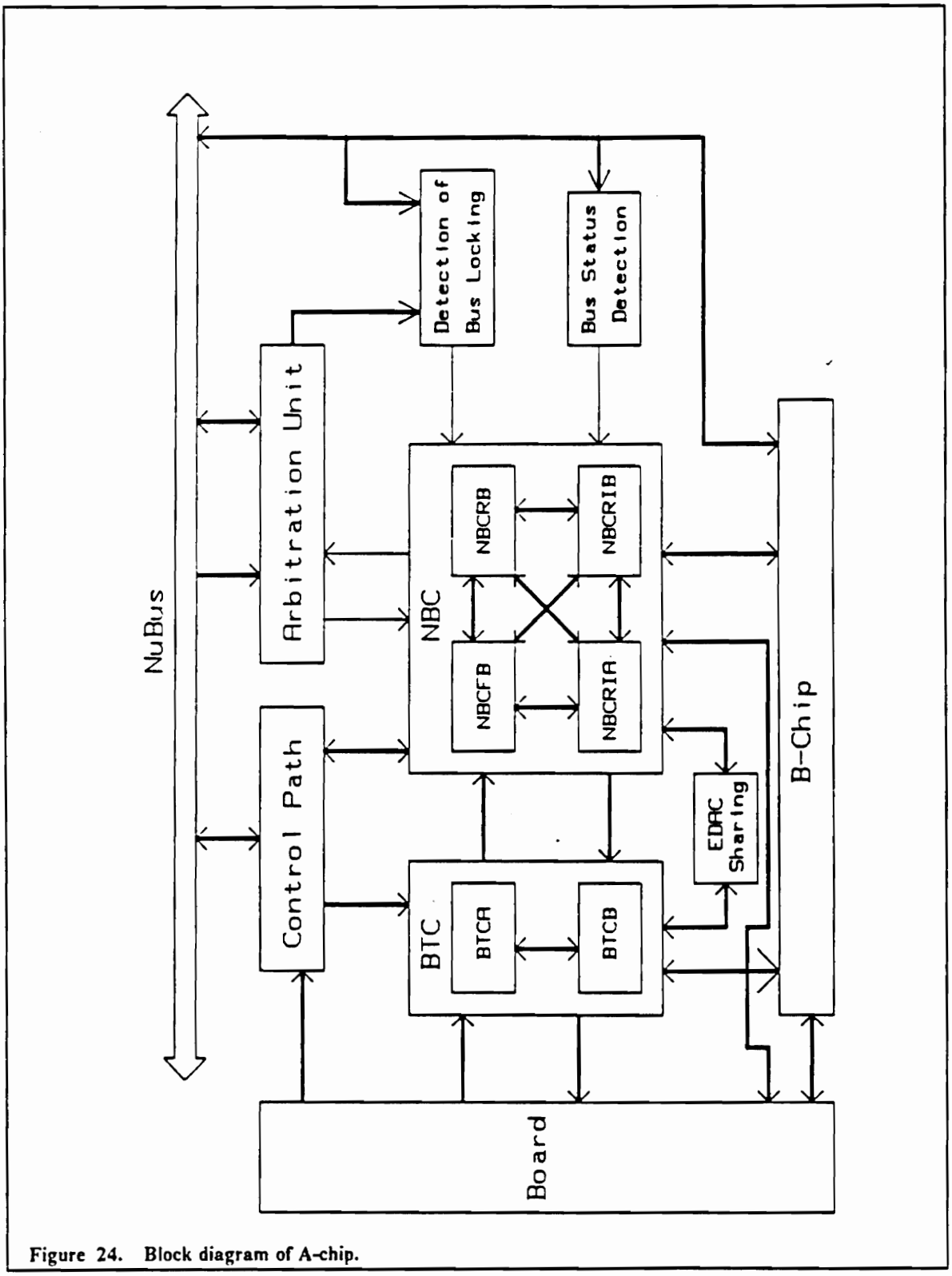


Figure 24. Block diagram of A-chip.

tration begins at the next rising edge of the bus clock. All modules that contest for the bus before this edge can contend for the bus. All other modules, even if they have higher priority, must wait until every module that requested for bus gets control of the bus and finishes its transaction.

If a module wins the arbitration contest, completes its transaction and does not want to lock the bus, it deasserts the RQST line. Then, the rest of the modules, with their RQST lines asserted, compete again for bus mastership. When the last module which requested the bus completes its transaction and deasserts the RQST line, RQST goes high since no module is pulling it low. Now, the modules, which could not participate in the previous round of arbitration (because the RQST line was already low), pull the RQST low and start an arbitration contest. With this arbitration scheme, a lower priority module can get control of the bus within a certain period of time, since a higher priority module cannot lock the bus all of the time.

As shown in Figure 25 on page 74 the arbitration module consists of the supporting logic such as flip-flops and logic gates and the arbitration logic itself. Whenever a module wants to contest for the bus mastership, it sets the IRQST signal (coming from the NBC). If the RQSTI (coming from NuFTbus) is high and IRQST goes high, then on the next rising clock pulse of the bus clock BUSCLK, the module pulls RQSTI low, makes COMP high, and starts the arbitration. If the RQSTI line is already low, IRQST is held high and the module waits for the previous bus masters to complete their transactions. The simulation is shown in Figure 26 on page 75.

The arbitration logic is explained below with subcircuits CGIA and ABARB shown in Figure 27 on page 76 and Figure 28 on page 77, respectively.

Subcircuit CGIA: CGIA places one bit of the 5-bit ID code of the module on two arbitration lines of the bus (one in each set). The output of CGIA is functionally an open drain. This is explained as follows. Since the open-drain cell in the library does not provide an output on the same cell to input from, a bidirectional cell along with two inverters are used in this design. When the output of the NAND gate of Fig. 27 is low, the tristate output buffer is enabled and therefore a low is

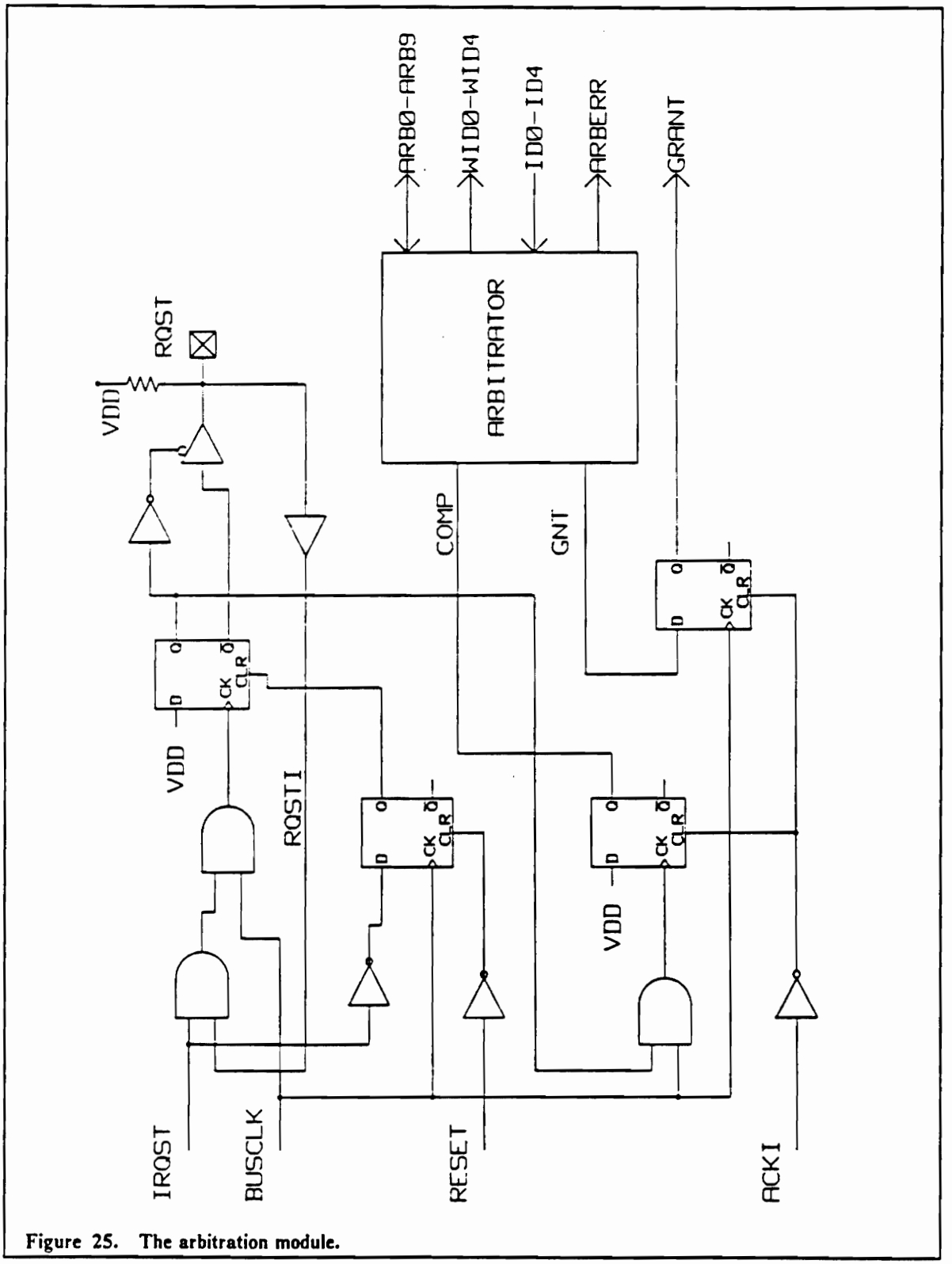


Figure 25. The arbitration module.

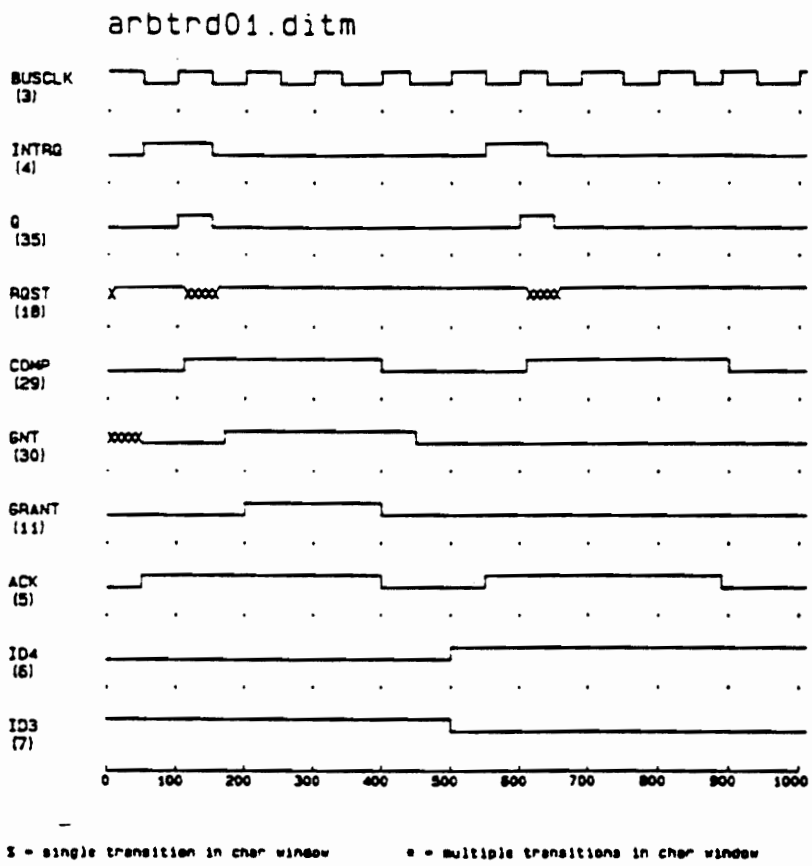


Figure 26. Simulation results for the arbitration module.

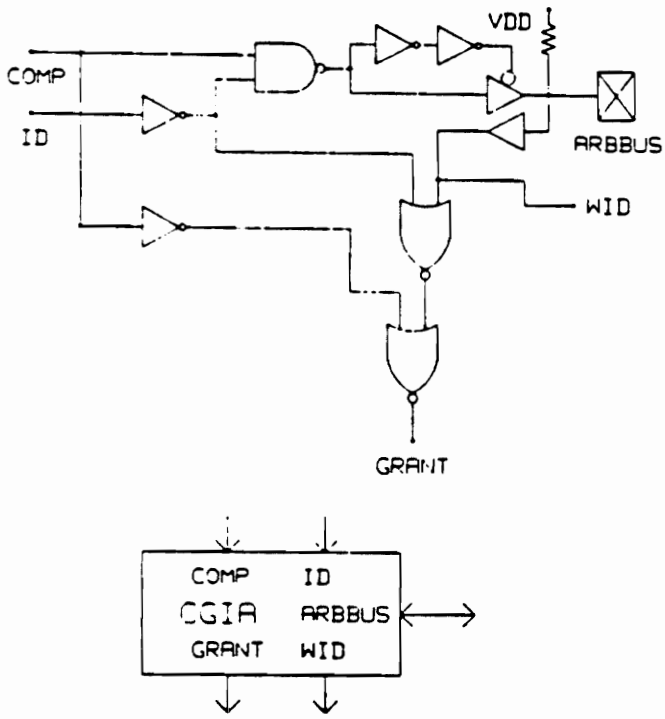


Figure 27. Subcircuit CGIA in the arbitrator.

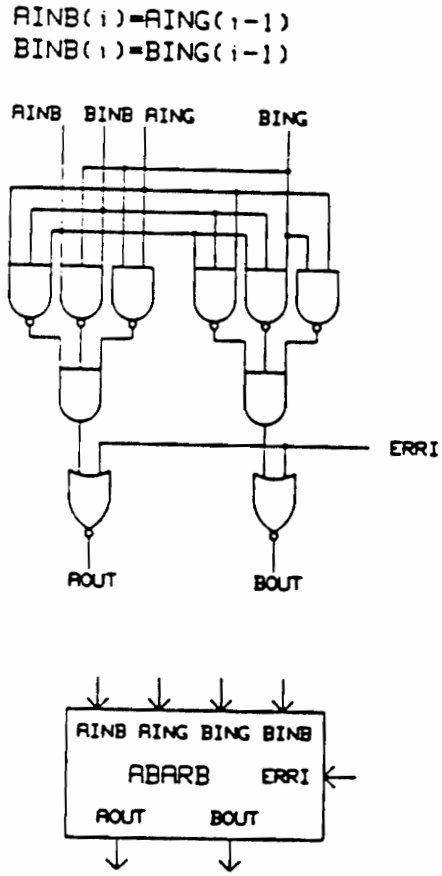


Figure 28. Subcircuit ABARB in the arbitrator.

present on the ARBBUS line. When the output of the NAND gate is high, a high impedance is present at the output of the output buffer (the two inverters in series are inserted because this macro cell does not allow its input signals to be shorted together). However, the input buffer is always enabled, if another module in the system has driven its ARBUS line low which is connected to this module, a low on the ARBBUS will be seen at the output of the input buffer. In this way, this configuration works as if it were an open drain gate. When a module wants to compete for bus mastership, it places a high on the COMP line. If this module also places a high on the ID line, and the ARBBUS line is still low, the CGIA realizes that some other module has placed a zero on its ID line and the CGIA deasserts its own grant signal, since "0" has higher priority than "1" in the arbitration contest. Further arbitration down through the low order bits of the ID code is then disabled from this unit. If the CGIA places a low on the ID line, it pulls the ARB line low and the GRANT signal is high, and therefore enables this module to continue with the arbitration at lower order bits. The WID signal is the winner ID code which will be used in detection of bus locking.

Subcircuit ABARB: The fault tolerant NuFTbus has two sets of arbitration lines. Arbitration is carried out on both sets at the same time. ABARB compares the results corresponding to a single ID line. If they both are equal, the arbitration proceeds to the next lower arbitration bit level. If they are different, the ABARB ignores these lines (bypassing a single bit error) and allows the arbitration to continue. However, if this is the second single bit error, then the ABARB disables the arbitration of the lower arbitration bit levels and asserts the ARBERR signal.

According to the NuFTbus specifications, bus arbitration should be settled within two bus cycles (200 ns.). As shown in the simulation plot in Figure 26 on page 75, the arbitration is started at the next rising edge after the INTRQ is asserted and GNT becomes high after 67 ns. At the next rising edge of the bus clock, GNT is latched and GRANT becomes high 100 ns. after the arbitration is started. The complete circuit with all the CGIAs and ABARBs connected together is shown in Figure 29 on page 80.

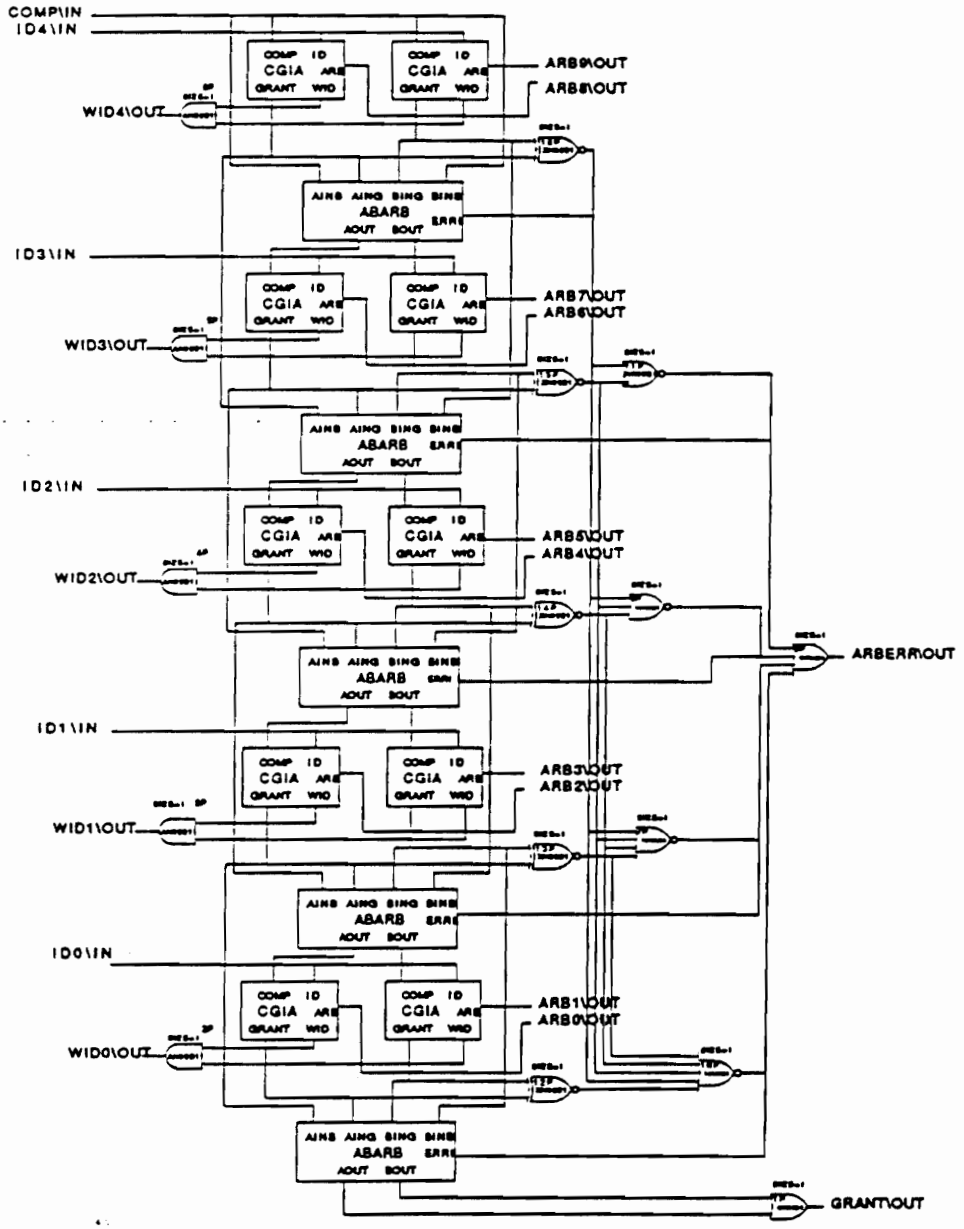


Figure 29. The arbitrator in the arbitration unit.

6.1.3 Board Transfer Controller And NuFTbus Controller

The controllers in the BIU, BTC and NBC, are designed by using the CAD tools PEG and LOGEN described in Chapter 4 and are partitioned into six smaller finite state machines so that they are small enough to be handled by the CAD tools. The six state machines are again pieced together to form the controller in the BIU, which is a FSM. The FSM consists of BTC and NBC. Going down the hierarchy, the BTC consists of BTCA and BTCB and the NBC consists of NBCRIA, NBCRIB, NBCRB and NBCFB. Their hierarchical relations are shown schematically in Figure 30 on page 82. The flow charts for BTCA, BTCB, NBCRIA and NBCRIB are shown starting with Figure 31 on page 83 through Figure 36 on page 88. And their designs are in the McLDL netlist form derived from PEG programs using the CAD tools described in chapter 5. State machine NBCRB is used to sample the signals from the NuFTbus and NBCFB is used to drive signals onto the NuFTbus. Their flow charts are not shown here. All simulations are based on FSM and the simulation is limited to 100 ns per step, i.e., simulation stimuli is changed every 100 ns., due to the limitation of the speeds UTMC cells can offer and the circuit complexity.

BTC and NBC are the control units used to control all of the other components of the BIU. For instance, the NBC sends signals to the arbitration module to cause it to arbitrate for mastership of the NuFTbus. After two bus cycles (200 ns., the arbitration cycle period), the NBC checks the GRANT signal from the arbitration module. A further example is that of the NBC or the BTC providing a latching pulse for the EDAC unit with CHK/GEN being either low or high depending on the status of AEPR or DEPR. After causing the EDAC unit to operate, the BTC and NBC check the DOUBLE ERROR output from the EDAC unit and enable the output register to put the address/data on the bus. All other modules in the BIU work with the controllers in a similar manner.

Each BIU (FSM) is simulated both as a master BIU and slave BIU corresponding to the four types of operation on the NuFTbus. These simulation are discussed in the following sections.

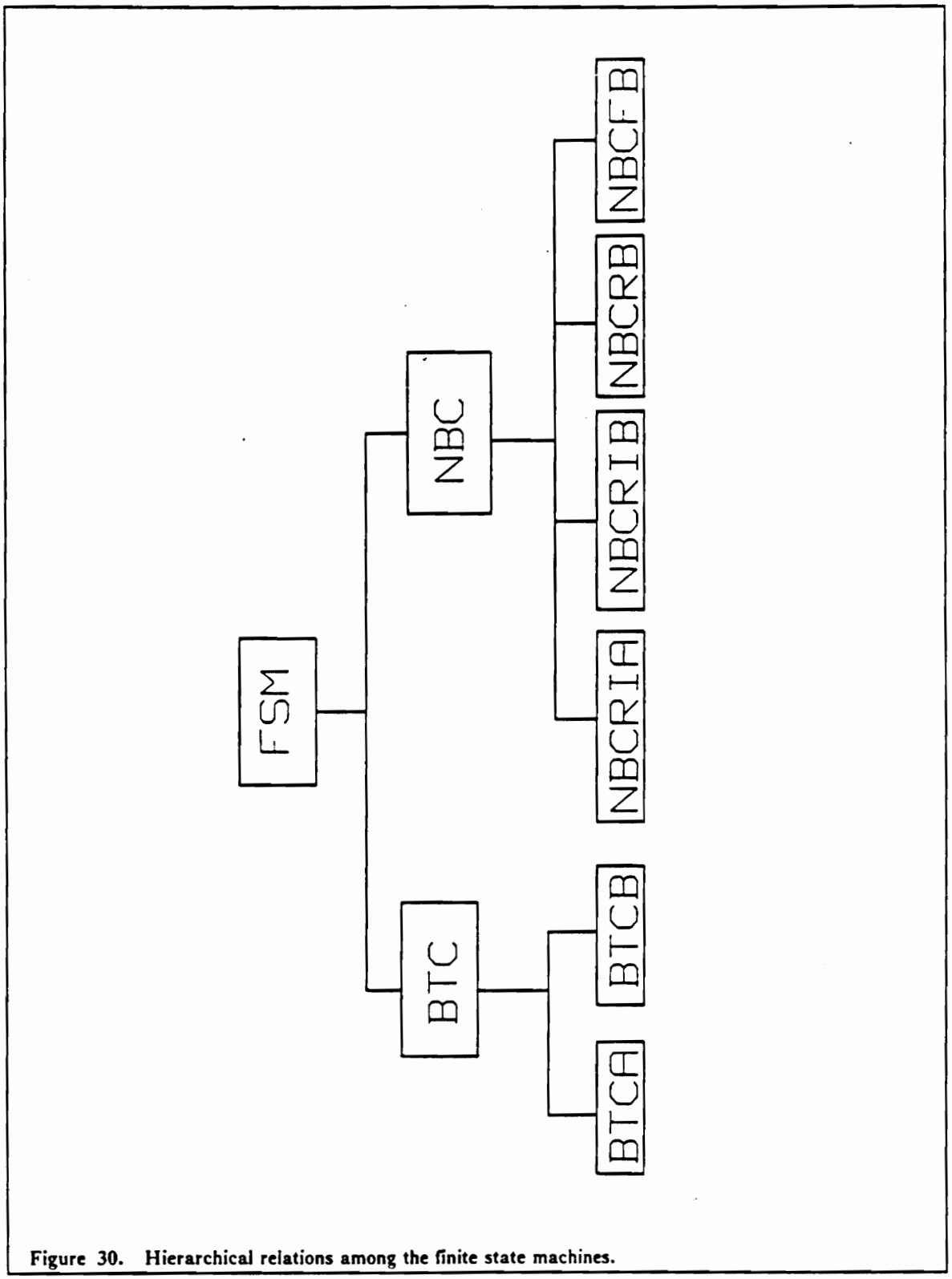


Figure 30. Hierarchical relations among the finite state machines.

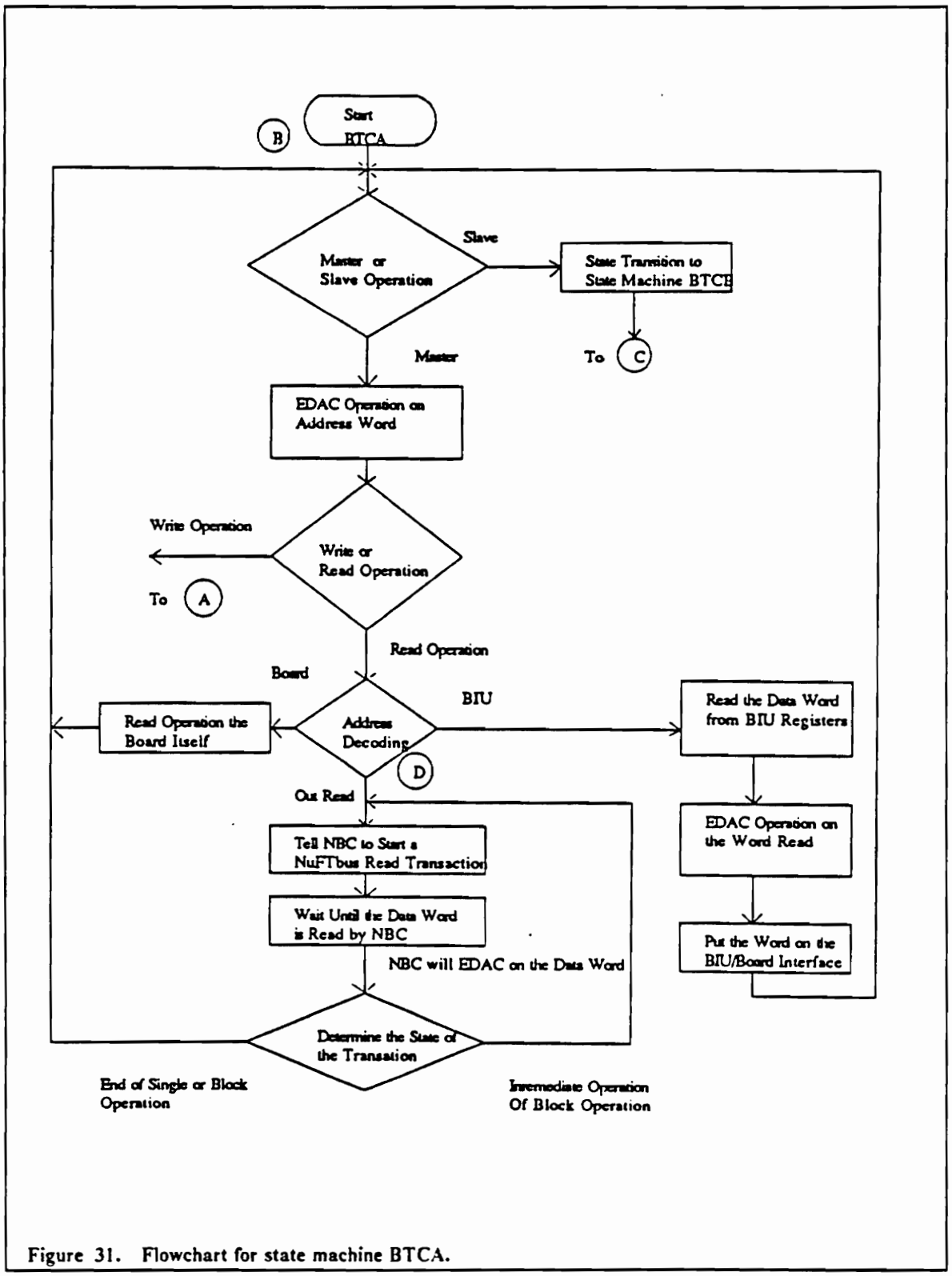


Figure 31. Flowchart for state machine BTCA.

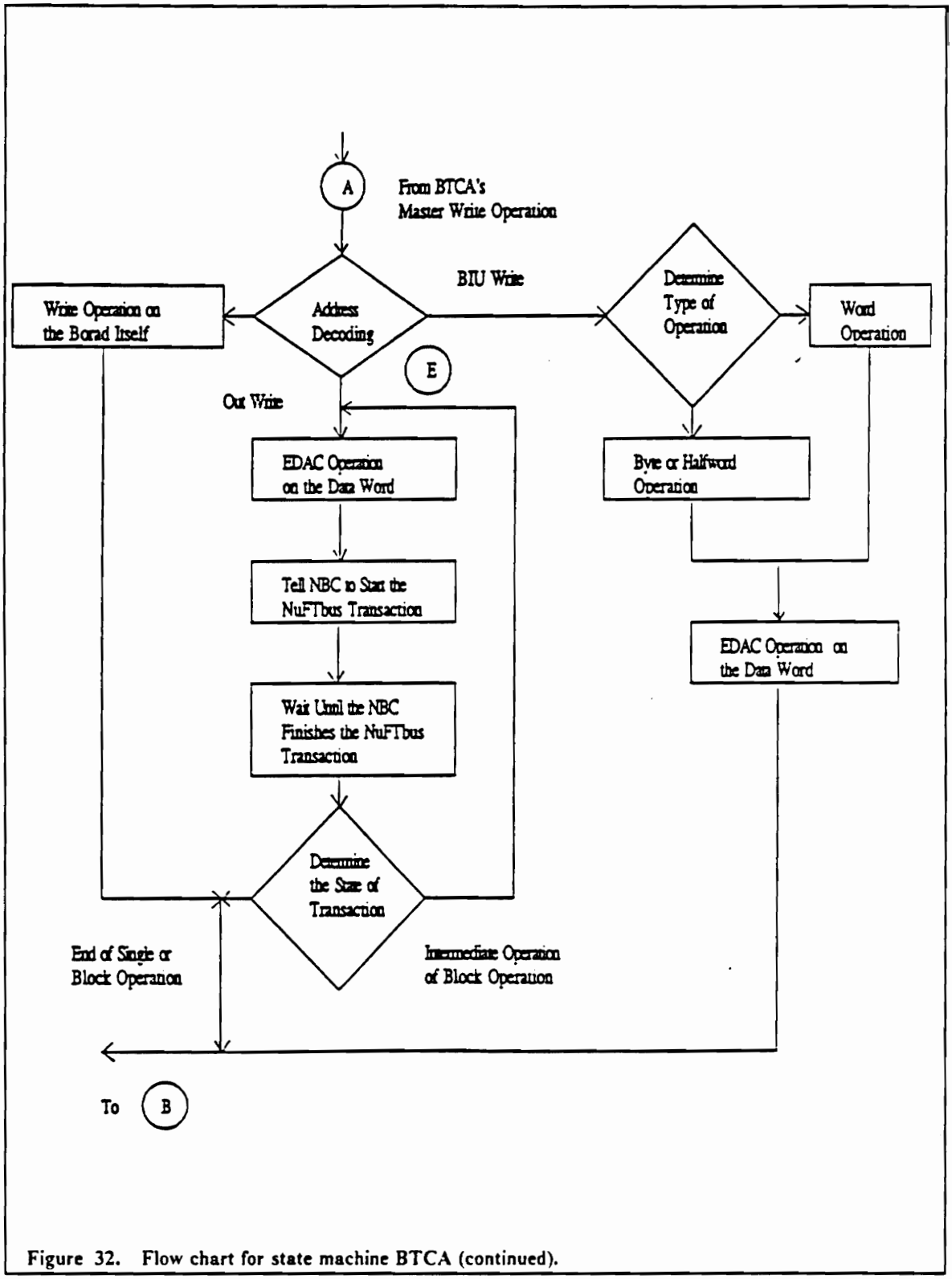


Figure 32. Flow chart for state machine BTCA (continued).

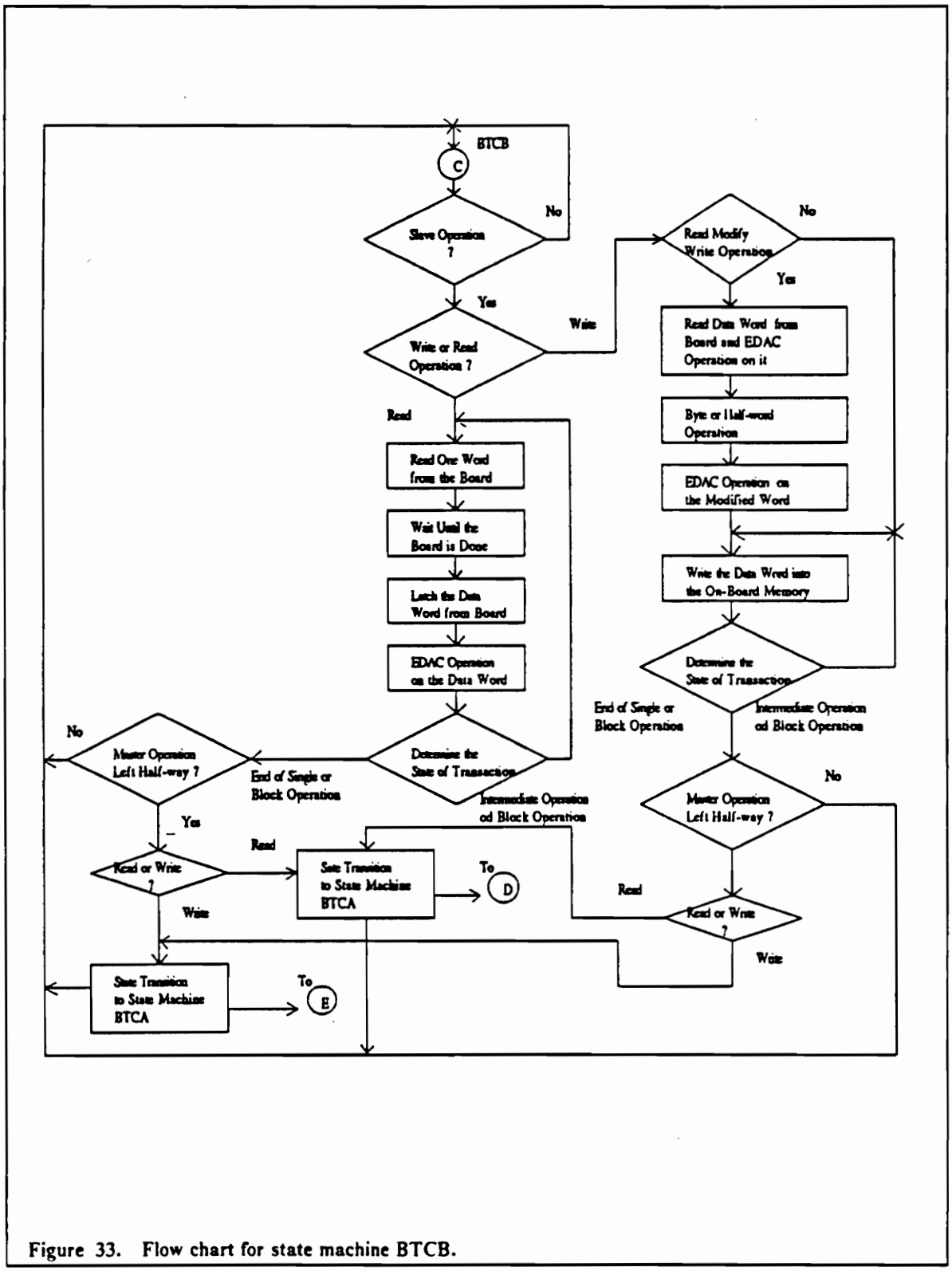


Figure 33. Flow chart for state machine BTCB.

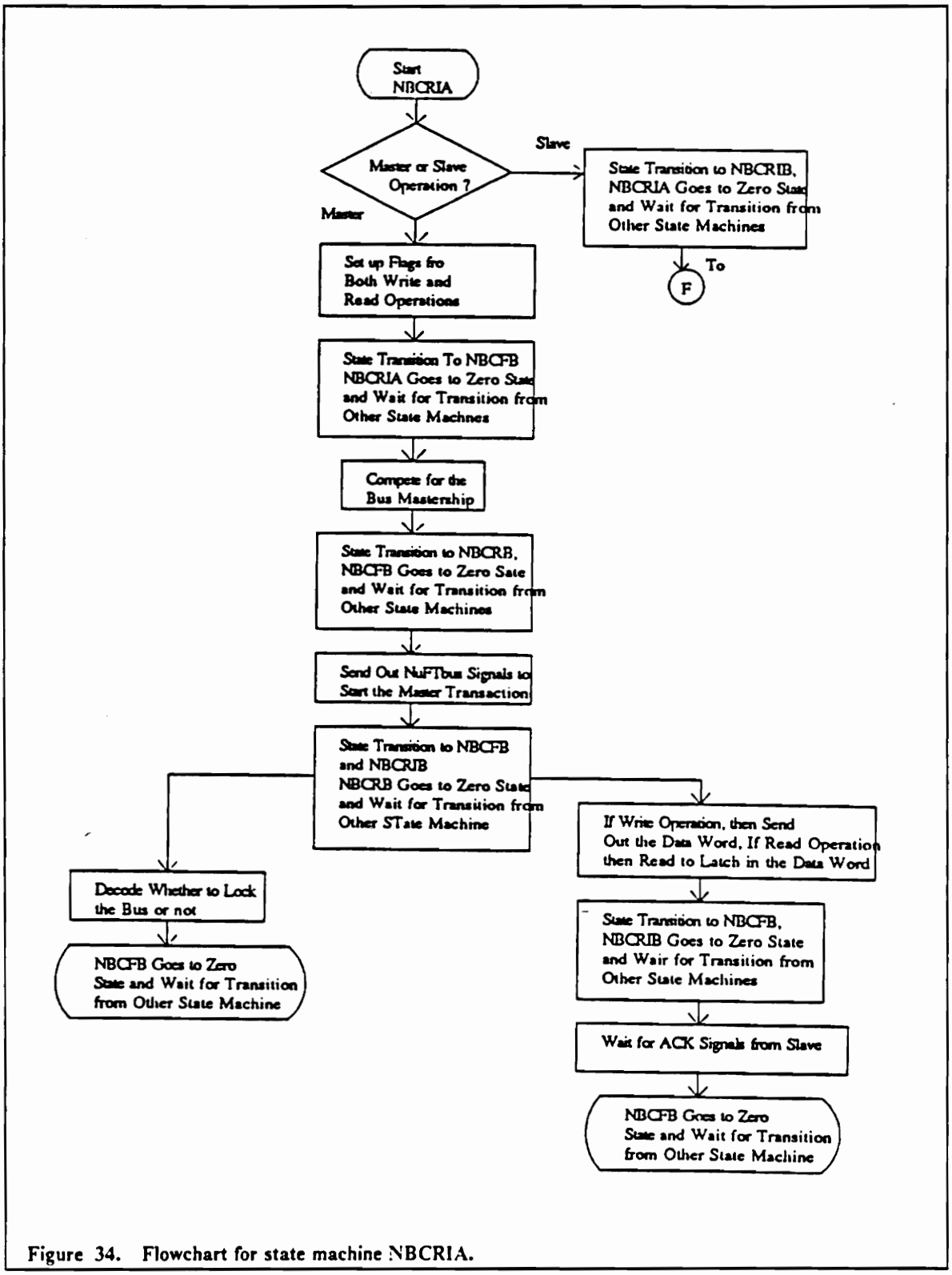


Figure 34. Flowchart for state machine NBCRIA.

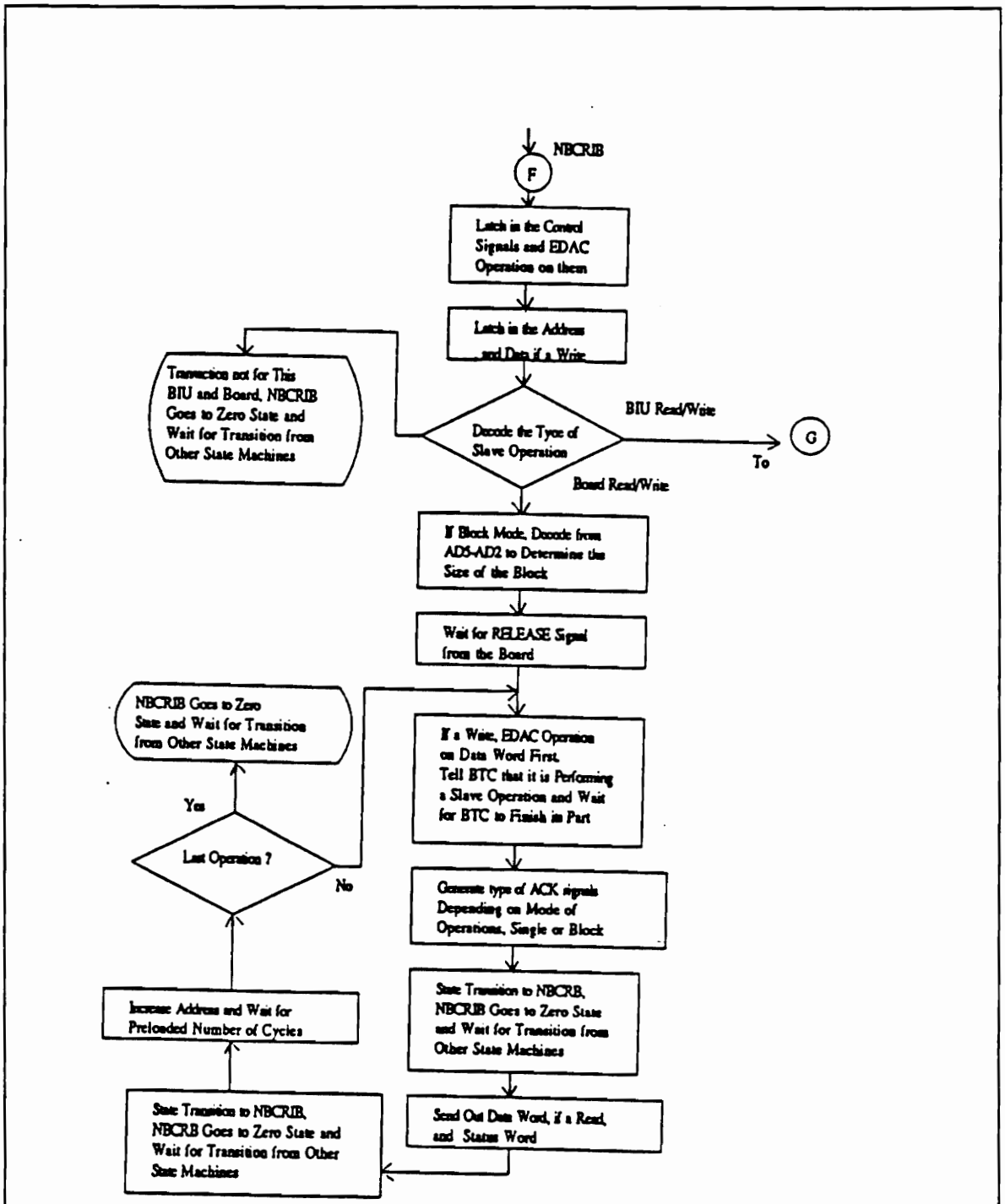


Figure 35. Flowchart for state machine NBCRIB.

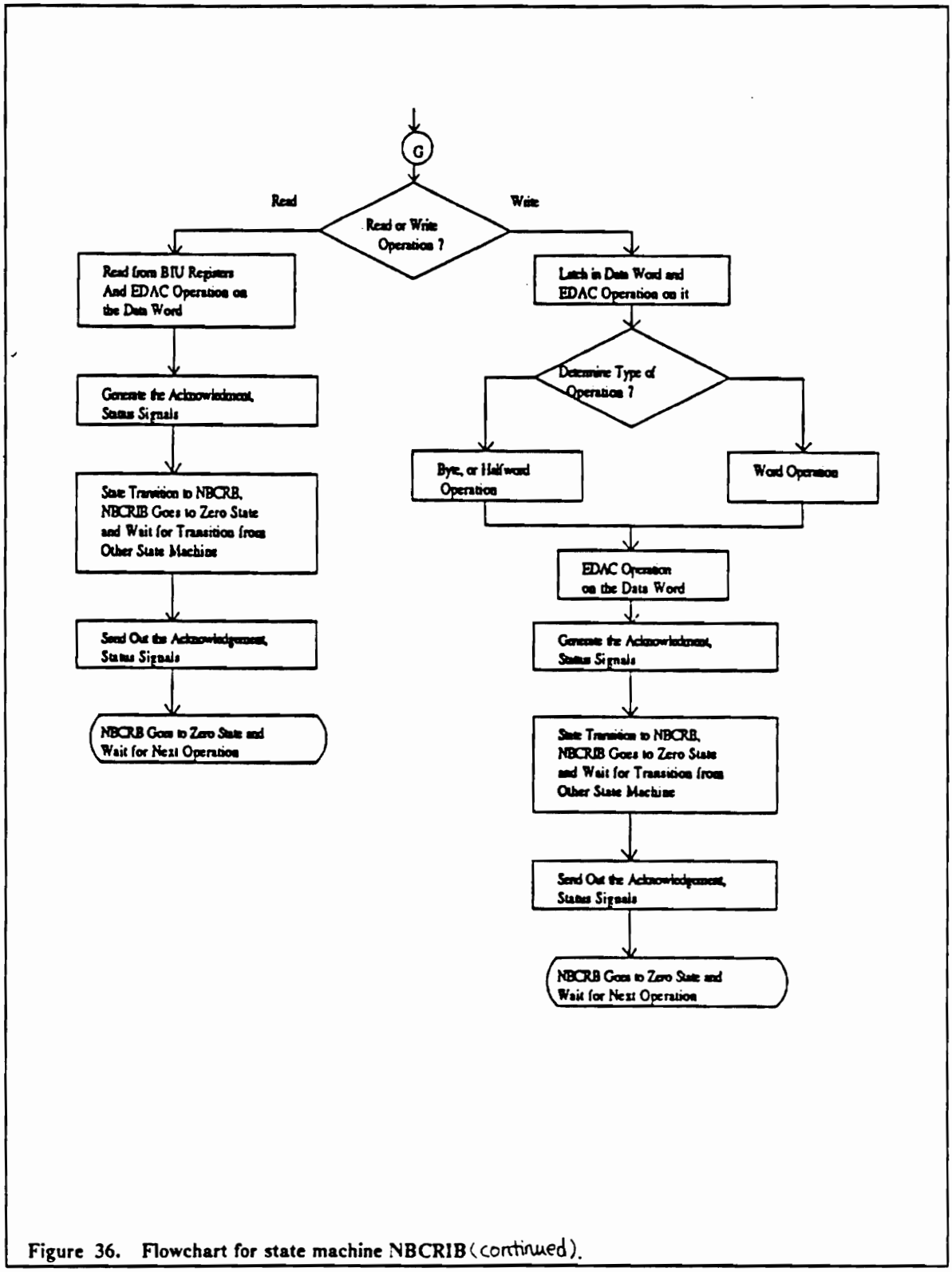


Figure 36. Flowchart for state machine NBCRIB (continued).

6.1.3.1 Master Side Single Read Operation: The input signals from the board and acknowledgement signals from the slave are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the master BIU when a single read operation is performed.

1. With a high on BLKEN (block enable) and a negative edge on SENI (single enable in), the BIU starts a single data transfer operation. A high on RWBI indicates that it is a read operation.

2. The BTC latches the address and the control information, such as AD_0, AD_1, Byte/Other, and RWBI, into appropriate registers.

3. The BIU decodes the address and determines that it should start a corresponding NuFTbus transaction to fetch data from the slave.

4. The BIU generates/checks the check bits for the address word depending on the AEPR (address EDAC provided) input.

5. The BTC puts the processor in a wait state by raising WAITO and raises NLOCK (NBC lock) to tell the NBC that a master mode operation is to be carried out.

6. The NBC sends an acknowledgement to BTC by raising NACK (NBC acknowledgement). Then BTC sets TOR (transfer out ready) to inform the NBC that it should continue the operation.

7. The NBC responds with a high on TIP (transfer in progress) and arbitrates for control of the NuFTbus. Once it obtains control of the NuFTbus it asserts the ISTART signal on the NuFTbus, outputs control information on the control lines, and the address on the address/data bus.

8. The NBC deasserts ISTART and the address on the next bus clock and waits for an acknowledgement from the slave.

9. When the ACKI input goes low, the NBC latches the status information and the data word. Using the EDAC units, the NBC determines the validity of the status lines and data word.

10. Once the NBC finishes fetching the data word from the slave, it raises DINRDY (data in ready). Then, the BTC puts the received data word on the board-BIU interface and deasserts WAITO.

11. SENI going high tells the BTC that the data has been accepted by the processor. So, the BTC lowers NLOCK to the NBC. In response, the NBC lowers its acknowledgement NACK and then both controllers go to their initial states waiting for the next operation.

The simulation results are shown in Figure 37 on page 90.

6.1.3.2 Slave Side Single Read Operation: The input signals from the board and from the master BIU are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the slave BIU when a single read operation is performed.

1. When the STARTI signal goes low on the NuFTbus, the potential slave NBC latches the address and control information and passes it through the EDAC units to check their validities.

2. The slave BIU decodes the address and determines that the address is in its address space.

3. The slave NBC asks for the release of the local processor bus by raising REL (release). Once it gets a RELACK (release acknowledgement), it sets BLOCK (BTC lock) to tell the slave BTC that a slave mode operation is to be carried out.

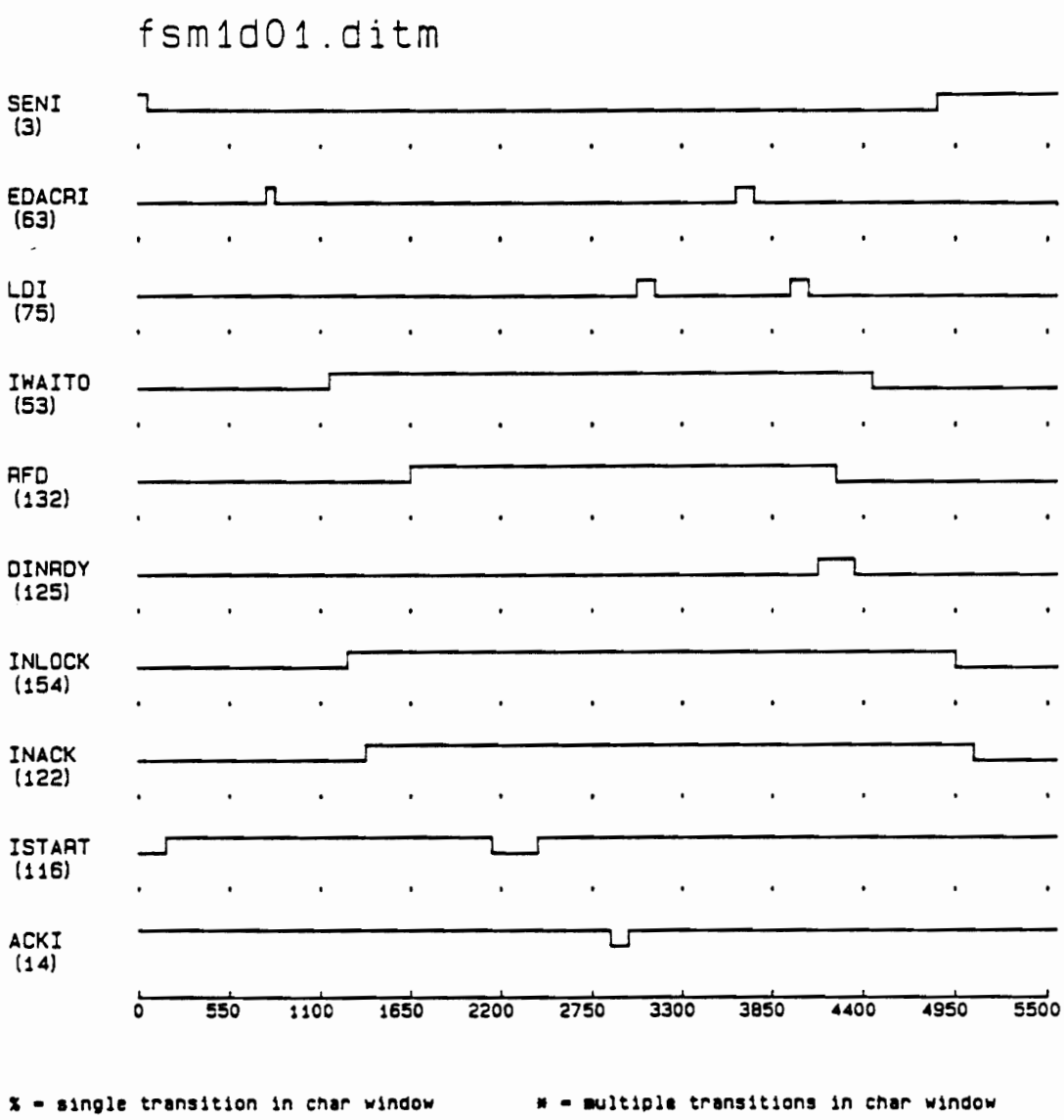


Figure 37. Master side single read operation.

4. The BTC sends an acknowledgement to NBC by raising BACK (BTC acknowledgement signal. With a high on BACK, the NBC set TIP and AIR (address in ready) signals.

5. The BTC places the received address on the board-BIU interface and lowers SENO. The processor places the data word on the data bus.

6. If the decode and read operations on the board are slow, WAITI goes high to tell the BTC to wait. Then the BTC latches the data word after the WAITI line goes low. In this simulation, WAITI is low all the time. So, the BTC sets SENO and latches the data input from the local bus into the Data Out Register.

7. The slave BTC checks the validity of the data word using the EDAC unit and sets TOR high. NBC waits for TOR to go low.

8. Then, the BTC lowers TOR and the NBC acknowledges this by lowering AIR and sends the data on the NuFTbus accompanied with IACK and status signals.

9. The NBC lowers REL, BLOCK and TIP signals while the BTC lowers the BACK, and then both BTC and NBC go to their initial states.

The simulation results are shown in Figure 38 on page 92.

6.1.3.3 Master Side Single Write Operation: The input signals from the board and acknowledgement signals from the slave are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the master BIU when a single write operation is performed.

1. With a high on BLKEN (block enable) and a negative going edge on SENI (single enable in), the BIU starts a single data transfer operation. A low on RWBI indicates that it is a write operation.

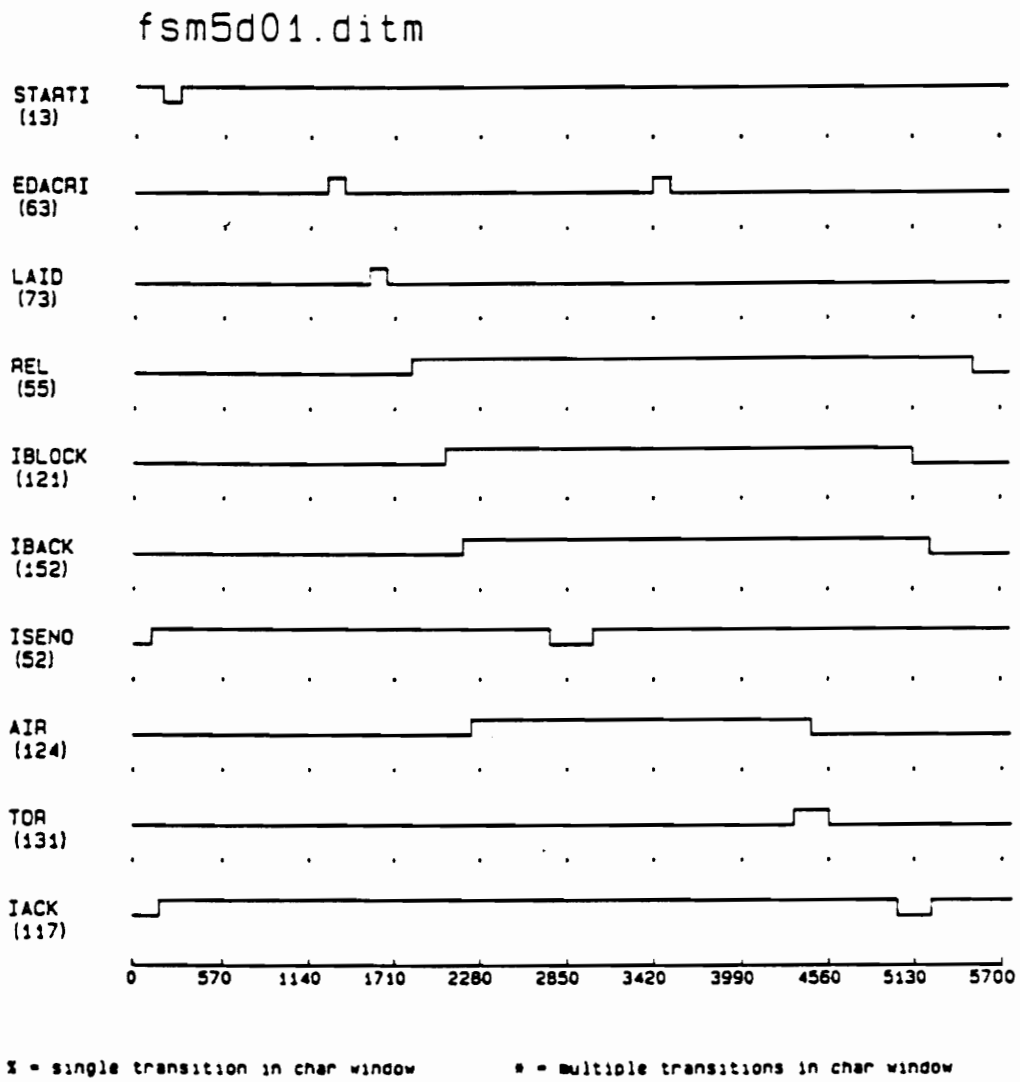


Figure 38. Slave side single read operation.

2. The BTC latches the address, data and the control information, such as AD_0, AD_1, Byte/Other, and RWBI, into appropriate registers.

3. The BIU decodes the address and determines that it should start a corresponding NuFTbus transaction to pass data to the slave.

4. The BIU generates/checks the check bits for the address word and data word depending on the AEPR (address EDAC provided) and DEPR (data EDAC provided) inputs.

5. The BTC puts the processor in a wait state by raising WAITO and raises NLOCK (NBC lock) to tell the NBC that a master mode operation is to be carried out.

6. The NBC sends an acknowledgement to BTC by raising NACK (NBC acknowledgement) signal. Then BTC sets TOR (transfer out ready) to inform the NBC that it should continue the operation.

7. The NBC responds with a high on TIP (transfer in progress) and AIR (address in ready), and arbitrates for control of the NuFTbus. The BTC lowers TOR and waits for AIR to go low. Once it obtains control of the NuFTbus it asserts the ISTART signal on the NuFTbus, output the control information on the control lines, and the address on the address/data bus.

8. The NBC deasserts ISTART and the address on the next bus clock, places the data on address/data bus, and waits for an acknowledgement from the slave.

9. When the ACKI input goes low, the NBC latches the status information. With the EDAC units, it determines the validity of the status.

10. Once the NBC finishes the NuFTbus transaction, it lowers AIR. Then, the BTC lowers the WAITO to the processor.

11. SENI going high indicates the end of the operation. So, the BTC lowers NLOCK to the NBC. In response, the NBC lowers its acknowledgement NACK and TIP, and then both controllers go to their initial states waiting for the next operation.

The simulation results are shown in Figure 39 on page 95.

6.1.3.4 Slave Side Single Write Operation: The input signals from the board and from the master BIU are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the slave BIU when a single write operation is performed.

1. When the STARTI signal goes low on the NuFTbus, the potential slave NBC latches the address and control information and passes it through the EDAC units to check its validities.

2. The slave BIU decodes the address and determines that the address is in its address space. Then it latches the data word from the NuFTbus and passes it through the EDAC unit.

3. The slave NBC asks for the release of the local processor bus by raising REL (release). Once it gets a RELACK (release acknowledgement), it sets BLOCK (BTC lock) to tell the slave BTC that a slave mode operation is to be carried out.

4. The BTC sends an acknowledgement to the NBC by raising the BACK (BTC acknowledgement signal. With a high on BACK, the NBC sets the TIP and AIR (address in ready) and DINRDY (data in ready) signals.

5. The BTC places the received address and data words on the board-BIU interface and lowers SENO.

6. If the decode and write operations on the board are slow, the WAITI goes high to tell the BTC to wait. The BTC waits until WAITI goes low and then sets SENO high. In this simulation, WAITI is low all the time.

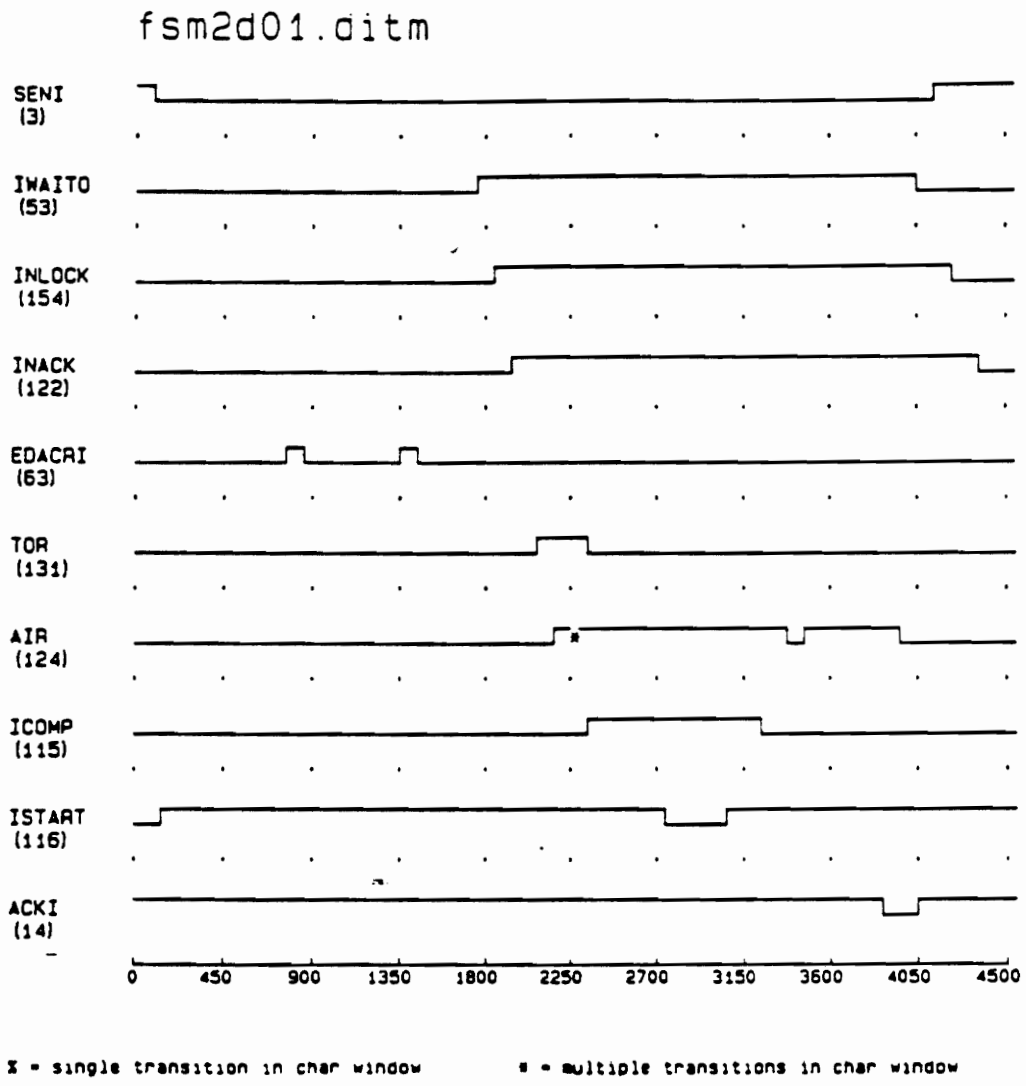


Figure 39. Master side single write operation.

7. The BTC raises TOR to indicate the end of the write operation. Then the NBC lowers AIR and DINRDY.

8. The BTC lowers TOR in response. Once the NBC sends out an acknowledgement on the NuFTbus, it lowers REL, BLOCK and TIP. Finally, the BTC lowers BACK, and both controllers go to their respective initial states.

The simulation results are shown in Figure 40 on page 97.

6.1.3.5 Master Side Block Read Operation: The input signals from the board and acknowledgement signals from the slave are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the master BIU when a block read operation is performed.

1. With a negative going edge on BLKEN (block enable) as well as on SENI (single enable in), the BIU starts a block data transfer operation. A high on RWBI indicates that it is a read operation.

2. The BTC latches the address and the control information, such as AD_0, AD_1, Byte/Other, and RWBI, into the appropriate registers.

3. The BIU decodes the address and determines that it should start a corresponding NuFTbus transaction to fetch data from the slave.

4. The BIU generates/checks the check bits for the address word depending on the AEPR (address EDAC provided) input.

5. The BTC puts the processor in a wait state by raising WAITO.

6. The BTC raises NLOCK (NBC lock) to tell the NBC that a master mode operation is to be carried out.

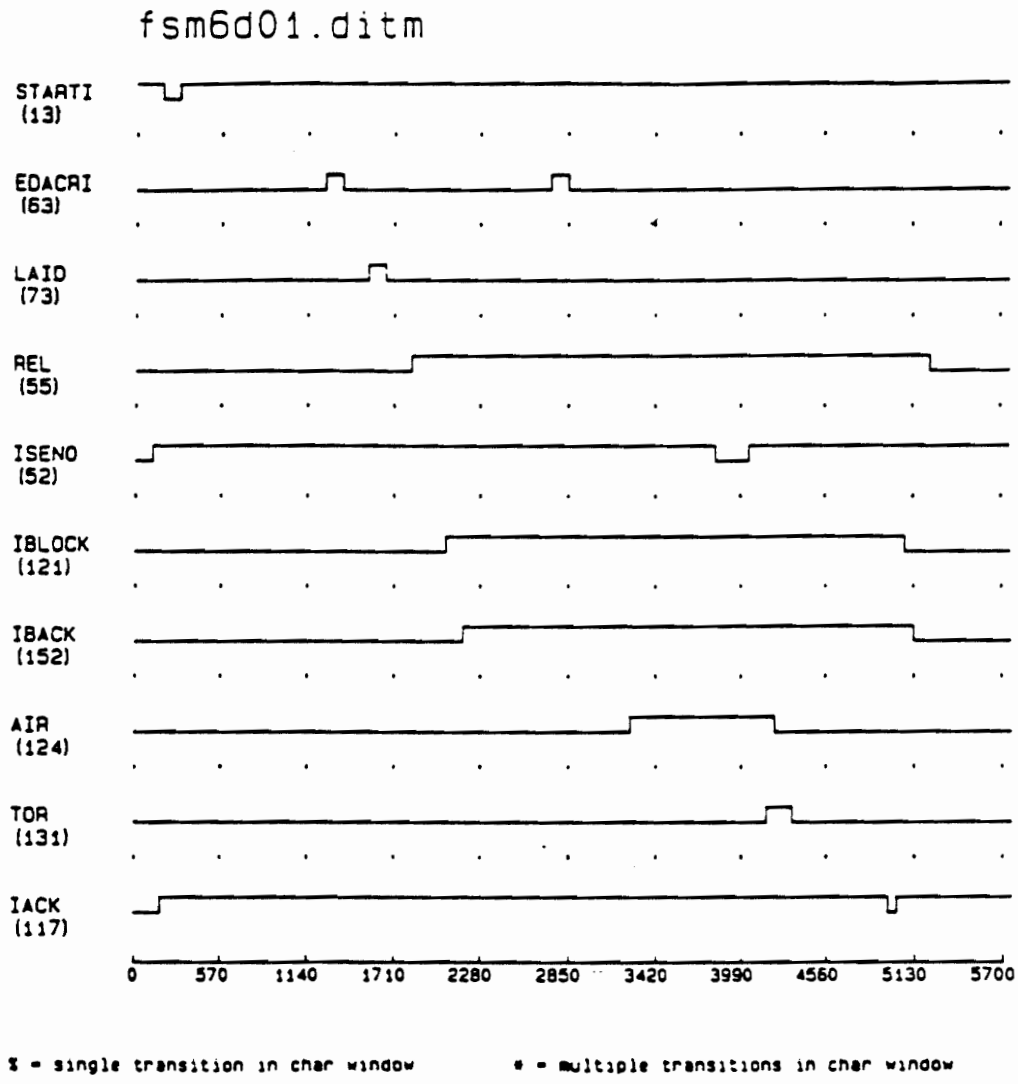


Figure 40. Slave side single write operation.

7. The NBC sends an acknowledgement to the BTC by raising the NACK (NBC acknowledgement) signal. Then BTC sets TOR (transfer out ready) to inform the NBC that it should continue the operation.

8. The NBC responds with a high on TIP (transfer in progress) and arbitrates for control of the NuFTbus. Once it obtains control of the NuFTbus it asserts the ISTART signal on the NuFTbus, output the control information on the control lines, and the address on the address/data bus.

9. The NBC deasserts ISTART and the address on the next bus clock and waits for an acknowledgement from the slave.

10. When the ACKI input goes low, the NBC latches the status information and the data word. With the EDAC units, it determines the validity of the status lines and data word.

11. Once the NBC finishes fetching the data word from the slave, it raises DINRDY (data in ready) to indica, the arrival of intermediate words on the NuFTbus. Then, the BTC puts the received data word on the board-BIU interface and deasserts the WAITO.

12. BLKEN going high with SENI still low indicates that the block transfer is not over. With BLKEN going low again, the BIU goes to step 5 and continues with the next word data transfer.

13. Both SENI and BLKEN going high indicate the end of block transfer and the BIU goes back to its initial state awaiting the next operation.

The simulation results are shown in Figure 41 on page 100.

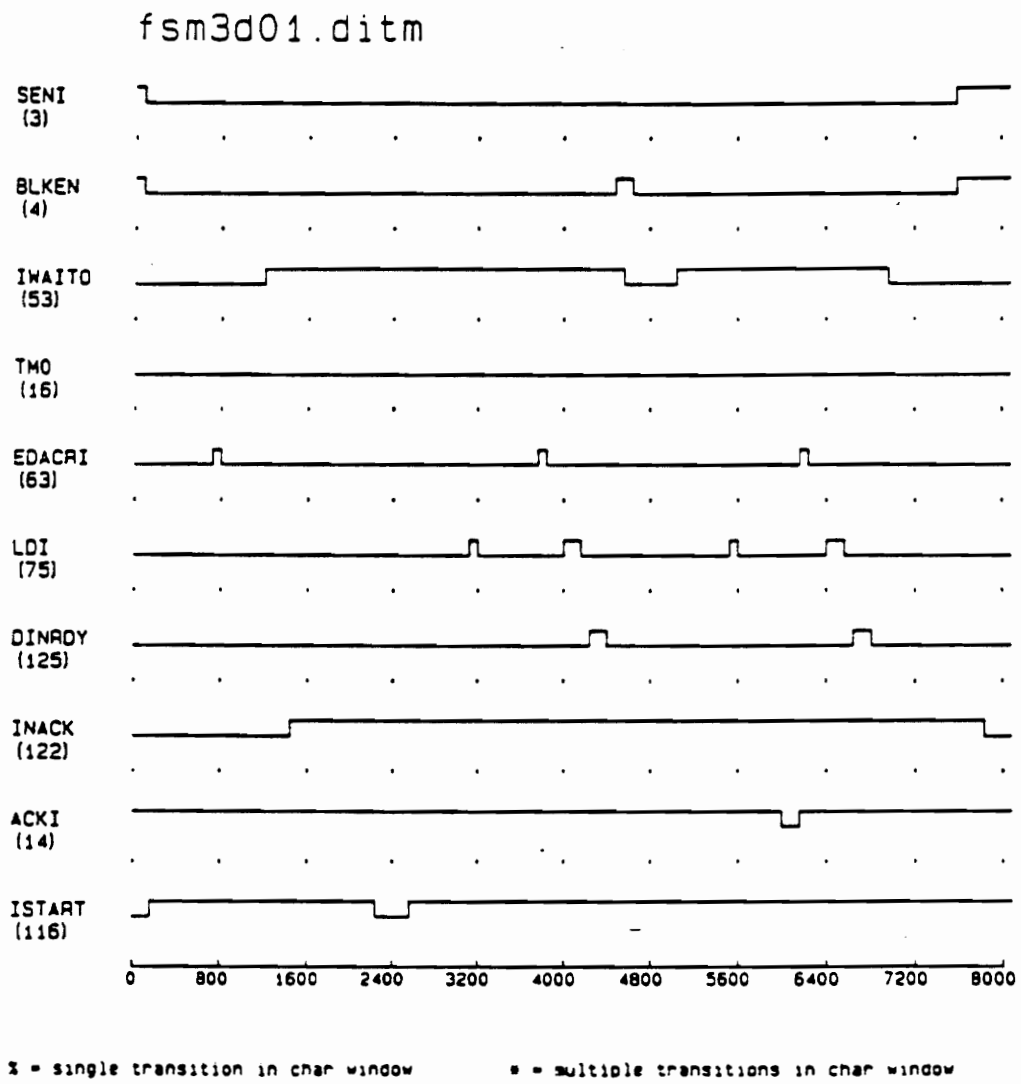


Figure 41. Master side block read operation.

6.1.3.6 Slave Side Block Read Operation: The input signals from the board and from the master BIU are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the slave BIU when a block read operation is performed.

1. When the STARTI signal goes low on the NuFTbus, the potential slave NBC latches the address and control information and passes it through the EDAC units to check its validity.

2. The slave BIU decodes the address and determines that the address is in its address space. The decoding of the control lines determines that a block read operation is to be performed.

3. The slave NBC asks for the release of the local processor bus by raising REL (release). Once the NBC gets a RELACK (release acknowledgement), it sets BLOCK (BTC lock) to tell the slave BTC that a slave mode operation is to be carried out.

4. The BTC sends an acknowledgement to the NBC by raising BACK (BTC acknowledgement) signal. With a high on BACK, the NBC sets the TIP and AIR (address in ready) signals.

5. The BTC places the received address on the board-BIU interface and lowers SENO. The processor places the data word on the data bus.

6. If the decode and read operation on the board are slow, the WAITI goes high to tell the BTC to wait. Then the BTC latches the data word after the WAITI line goes low. In this simulation, WAITI is low all the time. So, the BTC sets SENO and latches the data input from the local bus into the Data Out Register.

7. The slave BTC checks the validity of the data word using the EDAC unit and sets TOR high. NBC waits for TOR to go low.

8. The BTC lowers TOR and the NBC acknowledges this by lowering AIR and placing the data on the NuFTbus accompanied by the intermediate acknowledgement (TM_0).

9. On the next bus clock, the slave BIU deasserts TM_0 and tristates the data lines.

10. The slave BIU waits for n clock pulses before proceeding with the next data transfer (The number of clock cycles to wait is preprogrammed by the board designers or system architects).

11. The slave BIU decrements the word count, increments the address, puts the new address on the local processor bus and goes back to step 5.

12. After the transfer of the last data word, the slave BIU asserts IACK, instead of TM_0, to indicate the end of block operation.

13. Then, the NBC lowers REL, BLOCK and TIP signals while the BTC lowers the BACK, and then both of them go to their initial states.

The simulation results are shown in Figure 42 on page 102.

6.1.3.7 Master Side Block Write Operation: The input signals from the board and the acknowledgement signals from the slave are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the master BIU when a block write operation is performed.

1. With a negative going edge on SENI (single enable in) as well as on BLKEN (block enable), the BIU starts a block data transfer operation. A low on RWBI indicates that it is a write operation.

2. The BTC latches the address and control information, such as AD_0, AD_1, Byte/Other, and RWBI, into the appropriate registers.

3. The BIU decodes the address and determines that it should start a corresponding NuFTbus transaction to pass data to the slave.

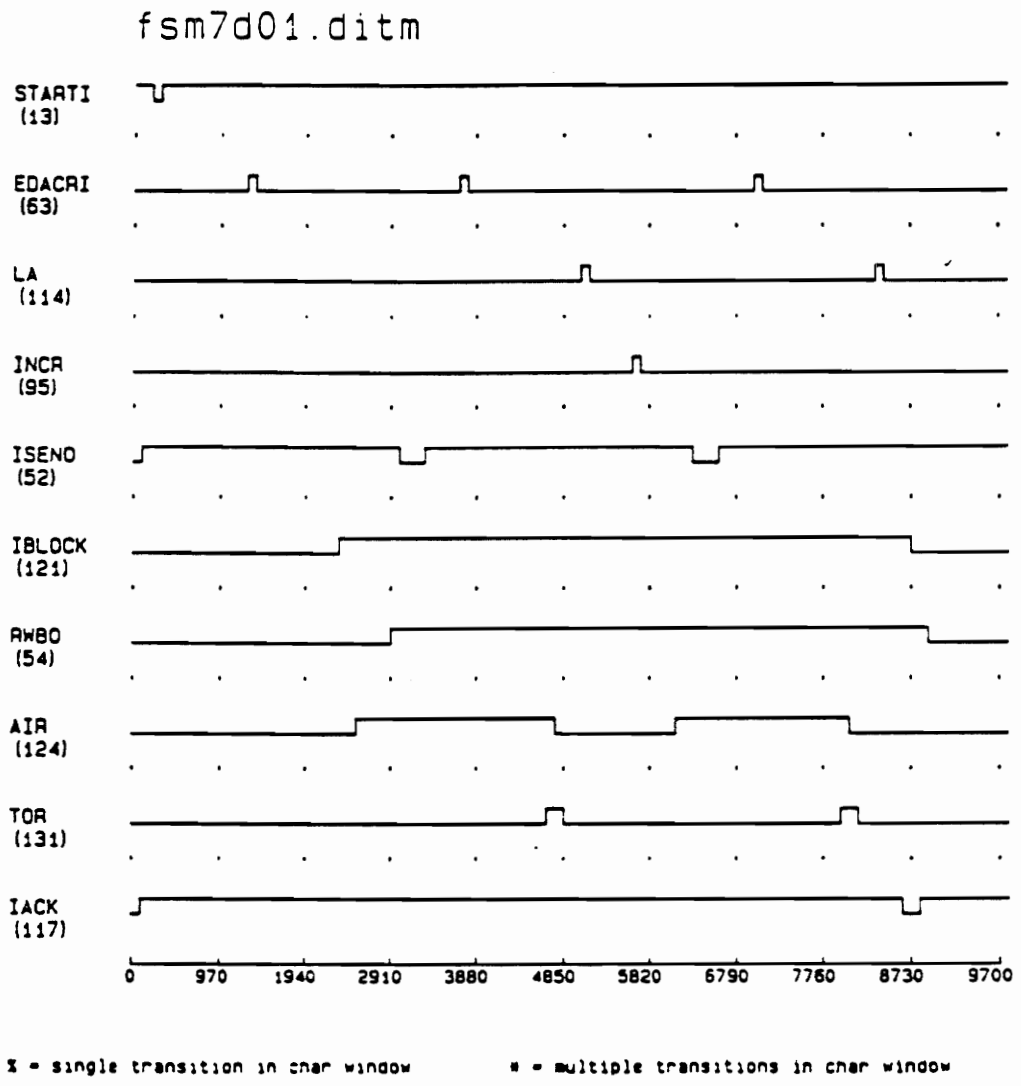


Figure 42. Slave side block read operation.

4. The BIU generates/checks the check bits for the address word and data word depending on the AEPR (address EDAC provided) and DEPR (data EDAC provided) inputs.

5. The BTC puts the processor in a wait state by raising WAITO.

6. The BTC raises NLOCK (NBC lock) to tell the NBC that a master mode operation is to be carried out.

7. The NBC sends an acknowledgement to the BTC by raising the NACK (NBC acknowledgement) signal. Then the BTC sets the TOR (transfer out ready) to inform the NBC that it should continue the operation.

8. The NBC responds with a high on TIP (transfer in progress) and AIR (address in ready), and arbitrates for control of the NuFTbus. The BTC lowers TOR and waits for AIR to go low. Once it obtains control of the NuFTbus it asserts the ISTART signal on the NuFTbus, outputs control information on the control lines, and the address on the address/data bus.

9. The NBC deasserts ISTART and the address on the next bus clock, places the data on the address/data bus, and waits for an intermediate acknowledgement from the slave.

10. With the intermediate acknowledgement TM_0 going low, the NBC deasserts the data lines and latches the status information. Using the EDAC unit, the NBC determines the validity of the status.

11. Once the NBC finishes one of the block NuFTbus transaction, it lowers AIR. Then, the BTC lowers the WAITO signal to the processor.

12. BLKEN going high with SENO still low indicates that the block transfer is not over. With BLKEN going low again, the BIU latches the next data word, goes to step 5, and continues with the next data transfer.

13. Both SENI and BLKEN going high indicates the end of the block transfer after which the BTC lowers the NLOCK signal to the NBC. In response, the NBC lowers its acknowledgement NACK and TIP, and then both controllers go to their initial states waiting for the next operation.

The simulation results are shown in Figure 43 on page 105.

6.1.3.8 Slave Side Block Write Operation: The input signals from board and from the master BIU are provided by the SIL simulation input file. The following is the sequence of steps that takes place in the slave BIU when a block write operation is performed.

1. When the STARTI signal goes low on the NuFTbus, the potential slave NBC latches the address and control information and passes it through the EDAC units to check its validities.

2. The slave BIU decodes the address and determines that the address is in its address space. Then it latches the data word from the NuFTbus and passes it through the EDAC unit. The decoding of the control lines determines that a block write operation is to be performed.

3. The slave NBC asks for the release of the local processor bus by raising REL (release). Once it gets a RELACK (release acknowledgement), it sets BLOCK (BTC lock) to tell the slave BTC that a slave mode operation is to be carried out.

4. The BTC sends an acknowledgement to NBC by raising the BACK (BTC acknowledgement) signal. With a high on BACK, the NBC sets the TIP, AIR (address in ready), { and DINRDY (data in ready) signals.

5. The BTC places the received address and data words on the board-BIU interface and lowers SENO.

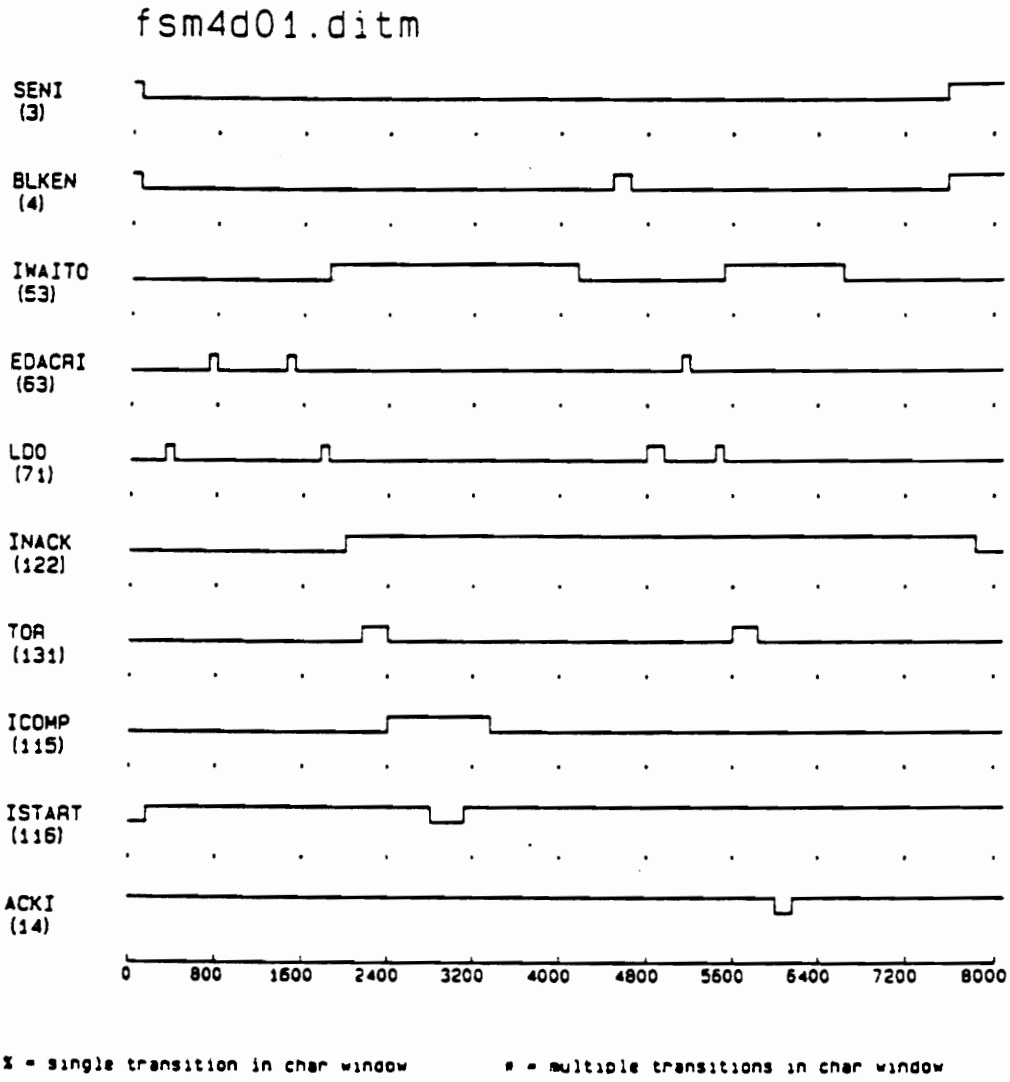


Figure 43. Master side block write operation.

6. If the decode and write operation on the board are slow the WAITI signal goes high to tell the BTC to wait. The BTC waits until WAITI goes low and then sets SENO high. In this simulation, WAITI is low all the time.

7. The slave BIU generates the necessary status signals and asserts an intermediate acknowledgement TM_0.

8. On the next bus clock, the slave BIU deasserts TM_0.

9. The slave BIU waits for n clock pulses before proceeding with the next data transfer (the number of clock cycles to wait is preprogrammed by board designers or system architects).

10. The slave BIU decrements the word count, increments the address, and latches the next data from the NuFTbus address/data lines. It passes the data through the EDAC unit.

11. The BTC raises TOR to indicates the end of one of the block write operations. Then the NBC lowers AIR and DINRDY. The BTC lowers TOR in response. If not end of the block operation then goes to step 5.

12. After the transfer of the last data word, the slave BIU asserts IACK, instead of TM_0, to indicate the end of block transfer. The NBC lowers REL, BLOCK, and TIP. Finally, the BTC lowers BACK, and both controllers go to their respective initial states.

The simulation results are shown in Figure 44 on page 107.

6.1.4 Control Path

The control path is used to generate and accept NuFTbus control signals: TM_1, $TM0_0$, AD_1, AD_0, As shown in Figure 45 on page 109, the control path consists of a 4-bit EDAC unit for

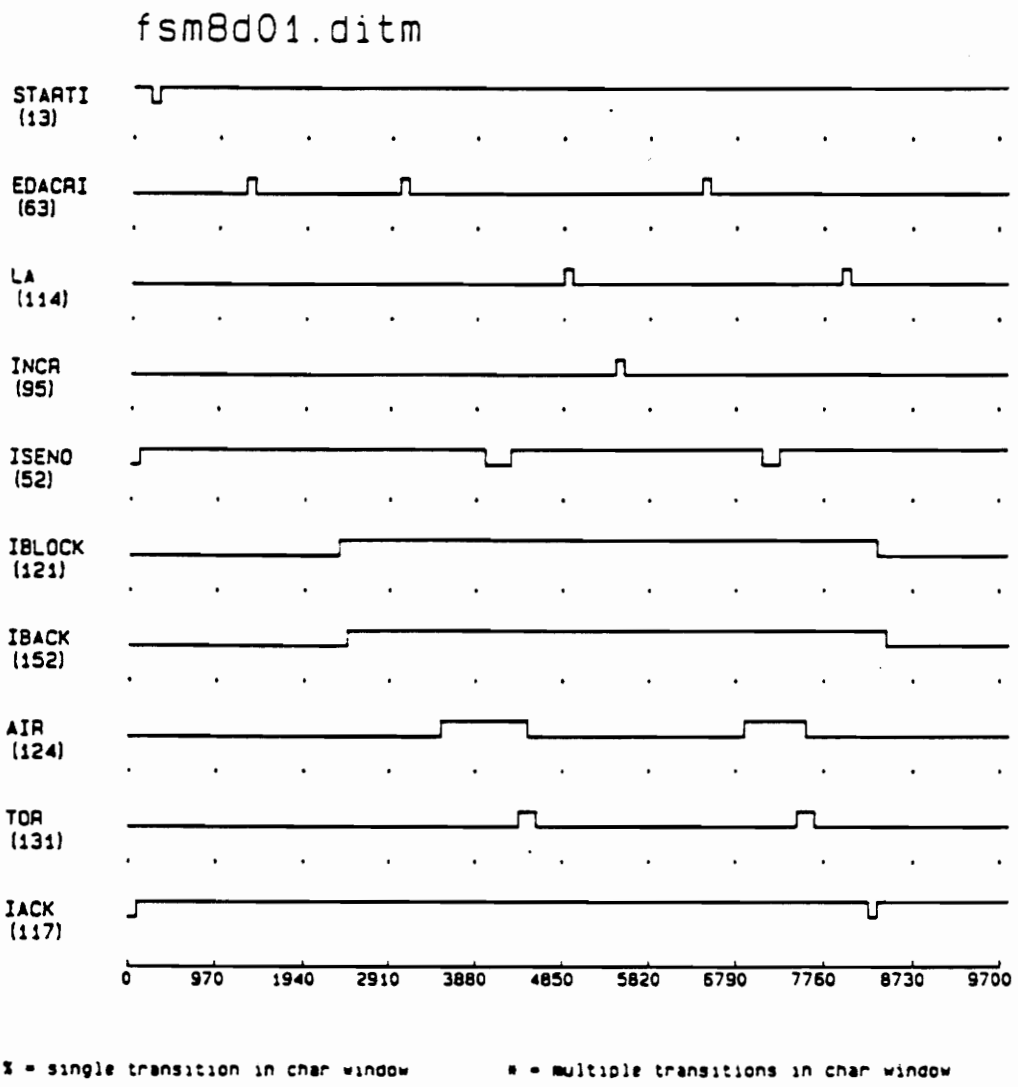


Figure 44. Slave side block write operation.

generating/checking the check bits for control lines, an 8-bit register with tri-state output for storing the control signals coming from the NuFTbus, and a 4-bit register with tri-state output to store control signals from the board. Additionally, the control path contains two 2-bit registers with tri-state output for generating the NuFTbus signals: ISTART, IACK, GTM_1, and GTM_0, two 3-to-8 decoders and one 2-to-4 decoder. The decoders are always enabled. The inputs to the decoders are provided from the direct outputs of the registers before tri-state buffers. This ensures that the control lines are always available in the decoded state, and moreover, those registers with tristate outputs can be tristated whenever necessary.

In the master mode, for a read/write operation, the control signals from the board RWBI, B/O*, AD_1, and AD_0, are latched into the 4-bit register using the LWSO (latch word size out) signal. If the operation is a write operation on one of the BIU registers, then B/O*, AD_1 and AD_0 are decoded to determine the word size; either byte, halfword or word. The NBC passes this control information to the EDAC unit using the pulses from WSOE and EDACRI2 to effect the transfer. After generating the check bits, the control signals are passed to the NuFTbus by enabling EDACOE2 and keeping RDN low. Acknowledgement signals from the slave module are latched from the NuFTbus by NLWSI. TM_1 and TM_0 are decoded by the 2-to-4 decoder to determine the type of acknowledgements. The acknowledgement may be TRFCOM (transfer complete), ERR (error), TMOUT (time out) or TRYLAT (try later).

In slave mode for a read/write operation, the signals on the NuFTbus are latched by the NBC using NLWSI (NuFTbus latch word size in). By enabling NLWSOE, RDN and EDACRI2, NuFTbus signals STARTI, ACKI, TM_1, and TM_0 are passed to the EDAC unit to be checked and subsequently passed back to the 8-bit register. The word size byte, halfword, or word is determined by using AD_1, AD_0, and TM_0 as inputs to the 3-to-8 decoder. At the end of the transaction, an acknowledgement is generated on the IACK, GTM_1, and GTM_0 lines. They are stored by the LA and LAS signals. and passed to the EDAC unit by AOE, SAOE and EDACRI2 signals. After generating the check bits, they are put on the NuFTbus by asserting EDACOE2 while keeping RDN low.

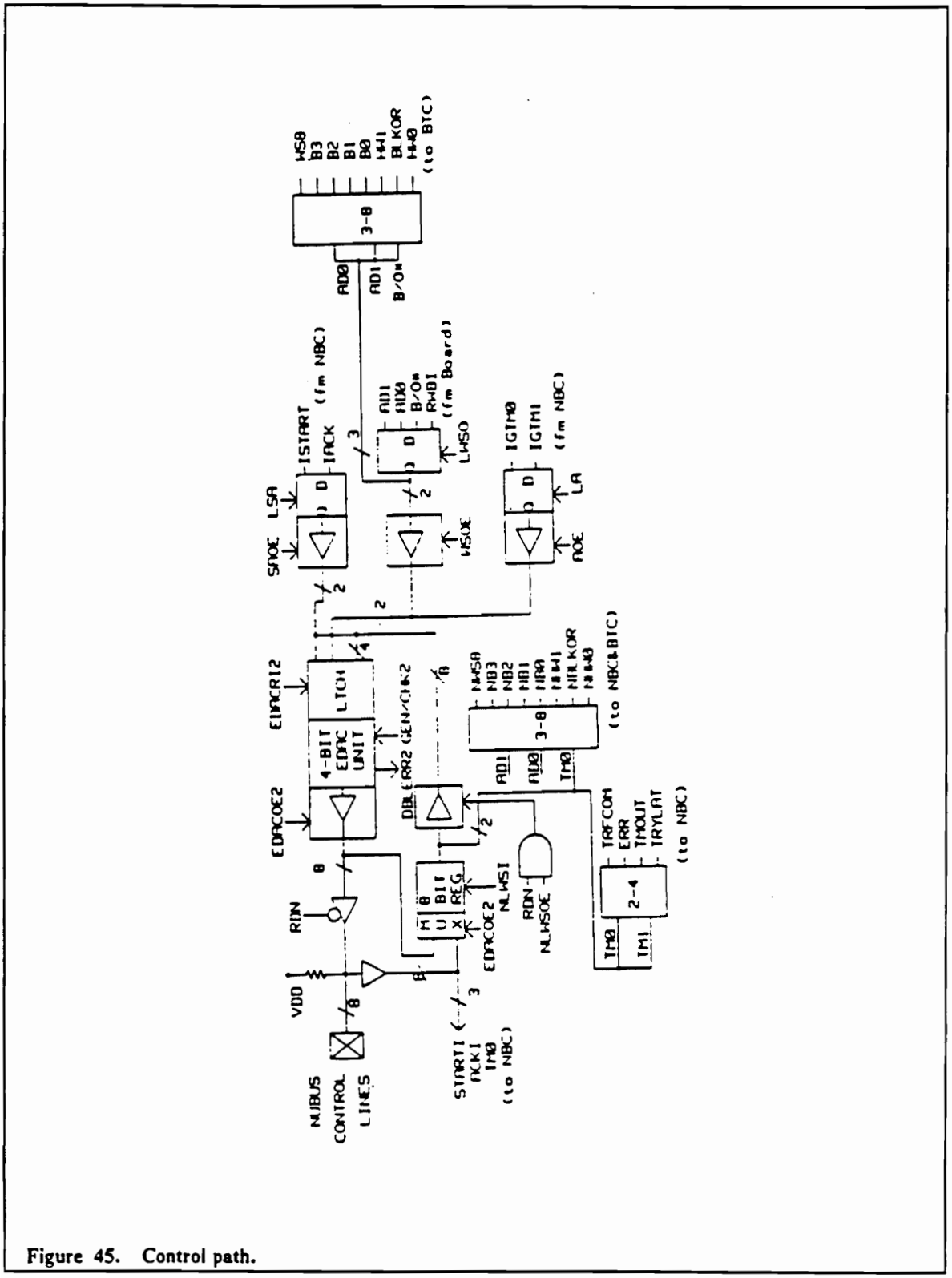


Figure 45. Control path.

The 4-bit EDAC and the 32-bit EDAC for the address/data word units for the control lines will be discussed further together in section 6.1.4.

6.1.5 Bus Status Detection

Since modules begin their bus mastership arbitration before the previous bus owner completes its NuFTbus transaction (if there is one performing a transaction), when a module wins the arbitration contest, it cannot immediately start a NuFTbus transaction until the bus is not busy. That is, the previous module which won the arbitration must first finish its operation. Therefore, the BIU has to keep track of the bus status, to see whether the bus is busy or available. This is done by the circuit shown in Figure 46 on page 111. A NuFTbus transaction is started by a high on ACKI and a low on STARTI, while a low on ACKI and a high on STARTI ends a transaction. So, with a high on ACKI and a low on STARTI, the flip-flop in the figure is set and the NBC sees a high on the BUSBUSY signal. Whenever ACKI goes low, it clears the flip-flop and the BUSBUSY becomes low. Once BUSBUSY becomes low, the BIU can start its NuFTbus transaction.

6.1.6 Detection of Bus Locking

Although modules normally perform only one transaction before allowing another requestor to become bus owner, sometimes a module may need to lock the bus. An example of this is an indivisible test-and-set operation performed in a multiprocessor environment. To lock the bus, a module simply continues to request bus mastership and participate in arbitration contests. Since it won the previous contest, and no other modules (possibly with higher priority) can join the contention, it will win subsequent contests.

The maximum number of transactions allowed in the lock mode is not stated in the NuFTbus specifications. It must then be decided in the system specifications. A 4-bit register is provided in

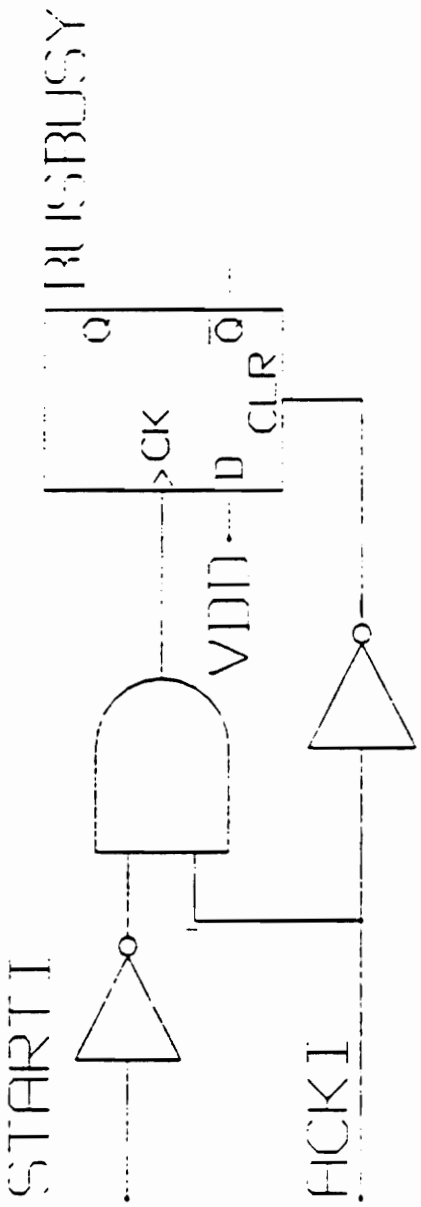


Figure 46. Bus status detection.

the BIU which is loaded with the maximum number of lock bus cycles allowed during initialization. Initially, both latches are reset, the output of the comparator is low, and the counter is reset. As shown in Figure 47 on page 113, the output of the comparator will go high only when the current winner ID code is different from the last winner ID code. With a falling edge on STARTI, the latch stores the current winner ID code coming from the arbitration module. This winner ID code is compared with that already stored in latch B. (For the very first transaction, this would be 0.) If the two ID codes are different, then the current winner ID code is passed to the latch B and the counter is reset. This winner ID code stored in latch B is compared with the winner ID code of the next transaction and so on. If the winner ID codes for two consecutive operations are the same due to bus locking, the counter is incremented at the rising edge of the STARTI signal. If the count matches the maximum number coming from the BIU register, a RSTBUS (restart bus) signal is generated and given to the NBC. The NBC then generates an attention cycle (STARTI and ACKI both low) for re-arbitration. The attention cycle also resets the counter.

6.1.7 EDAC Sharing Interface

Since the 32-bit EDAC unit is quite complicated and needs many logic gates for its implementation, the BTC and NBC share a common 32-bit EDAC unit for address and data word checking. A further discussion of the 32-bit EDAC unit can be found in Chapter 7). Therefore there is a need for a circuit and signals to control access to the EDAC. This circuit is shown in Figure 48 on page 114.

Whenever the BTC needs the EDAC unit, it set CHKEDAC line and on the next clock pulse, it checks the EDACBUSY1 line. If it is low, the BTC sets ACQEDAC1 line and gets control of the EDAC unit. Whenever the NBC needs the EDAC unit, it checks the state of the EDACBUSY2 line. If it is low, then the NBC sets the ACQEDAC2 line and gets control of the EDAC unit. When the BTC and NBC want to access the EDAC unit at the same time, higher priority is given to the BTC. More detailed descriptions of the handshaking are listed below.

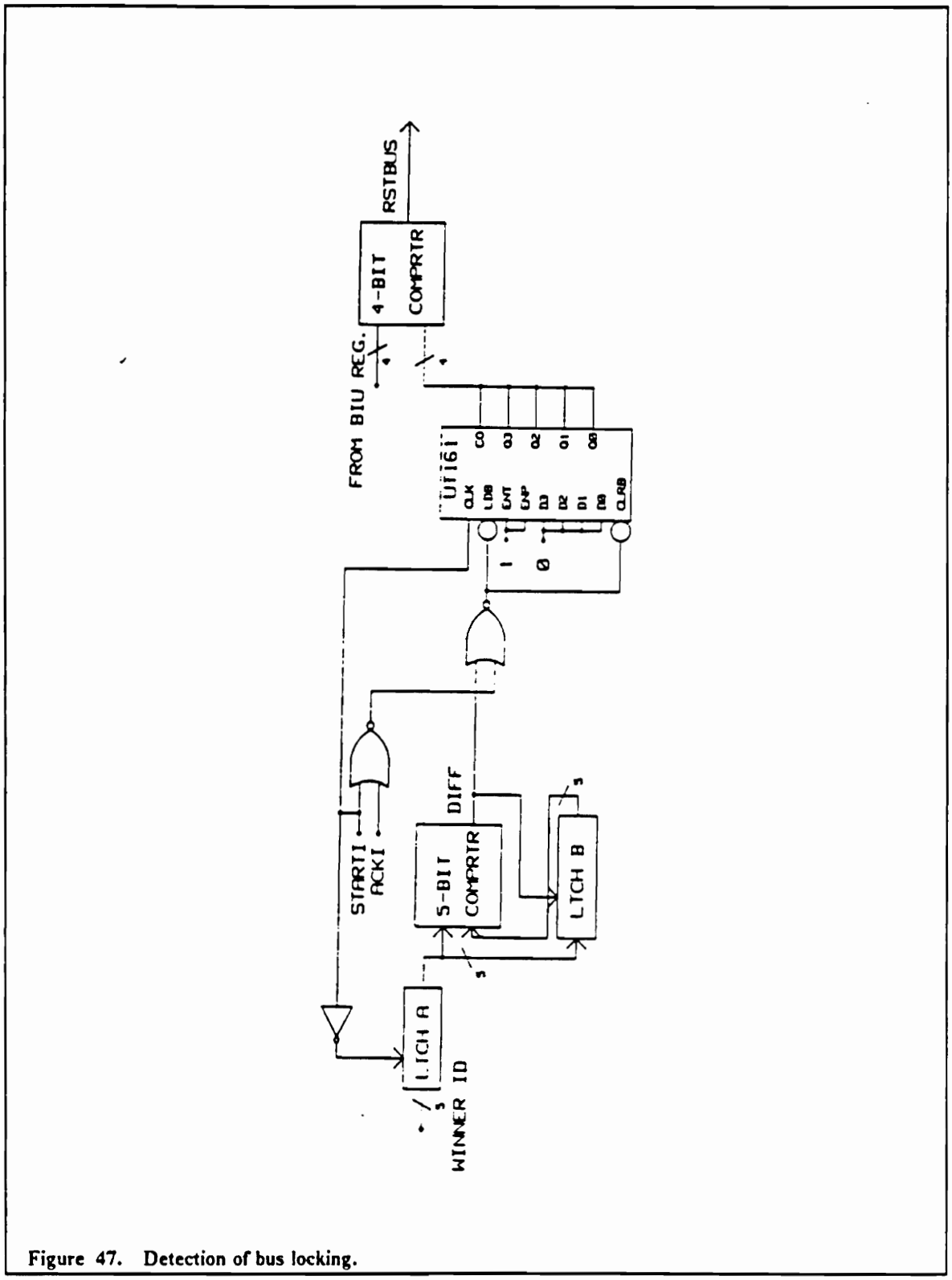


Figure 47. Detection of bus locking.

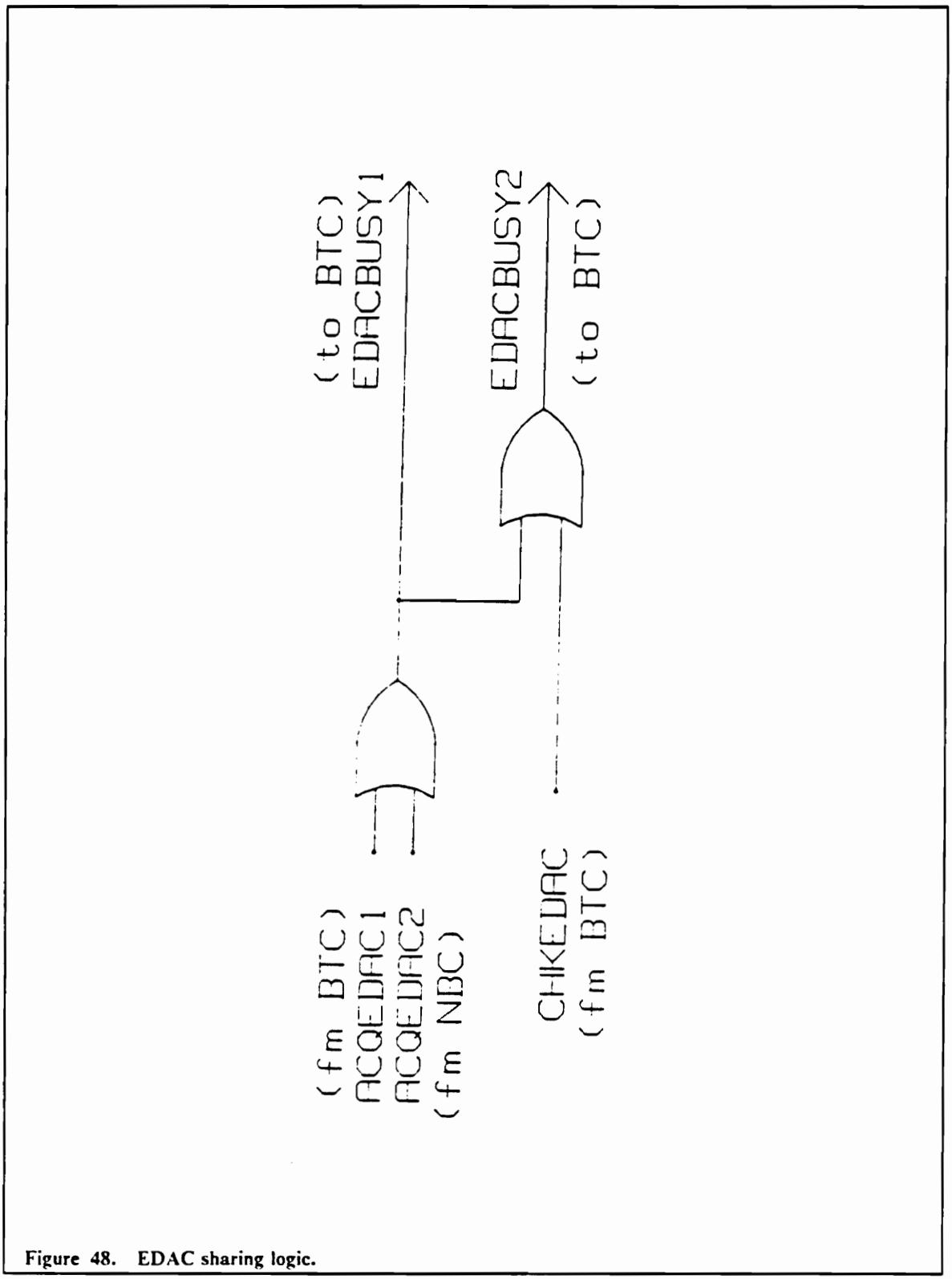


Figure 48. EDAC sharing logic.

- **EDAC unit available**: The BTC (NBC) checks the EDACBUSY1 (EDAC line. If it is low, the BTC (NBC) sets the ACQEDAC1 (ACQEDAC2) line and gets control of the EDAC unit.

- **EDAC unit busy**: The BTC (NBC) waits till the EDAc unit become available, and then gets control of it by setting the ACQEDAC1 (ACQEDAC2) line.

- **Both the BTC and NBC need the EDAC unit**: A problem would arise if the EDAC unit was available and both the BTC and the NBC checked the EDACBUSY lines at the same time. Both would get a low on the EDACBUSY lines, and therefore both would assume that they could get the EDAC unit. This is avoided by adding the CHKEDAC signal from the BTC which gives the BTC higher priority to access the EDAC unit. The BTC sets CHKEDAC one cycle before it checks the EDACBUSY1 line. By doing so, even if the BTC and the NBC check their respective EDACBUSY lines at the same time, the NBC will get EDACBUSY2 high (due to CHKEDAC), even if actually the EDAC unit is available at that time. The BTC will get control of the EDAC unit. Once the EDAC completes its operation, the NBC can then access it. Therefore, the addition of the CHKEDAC line solves the problem that both BTC and NBC want to access the EDAC unit at the same time.

6.2 B-Chip

6.2.1 The Block Diagram of B-Chip

As shown in Figure 49 on page 117 and Figure 50 on page 118, this chip consists of the following circuits: data path, address decode unit, BIU registers, 32-bit EDAC unit, block starting address detection, block operation address incrementer, block size decrementer, timer to wait between transfer in block mode operation, and the retry counting circuit. Each unit will be explained more in depth in the following sections.

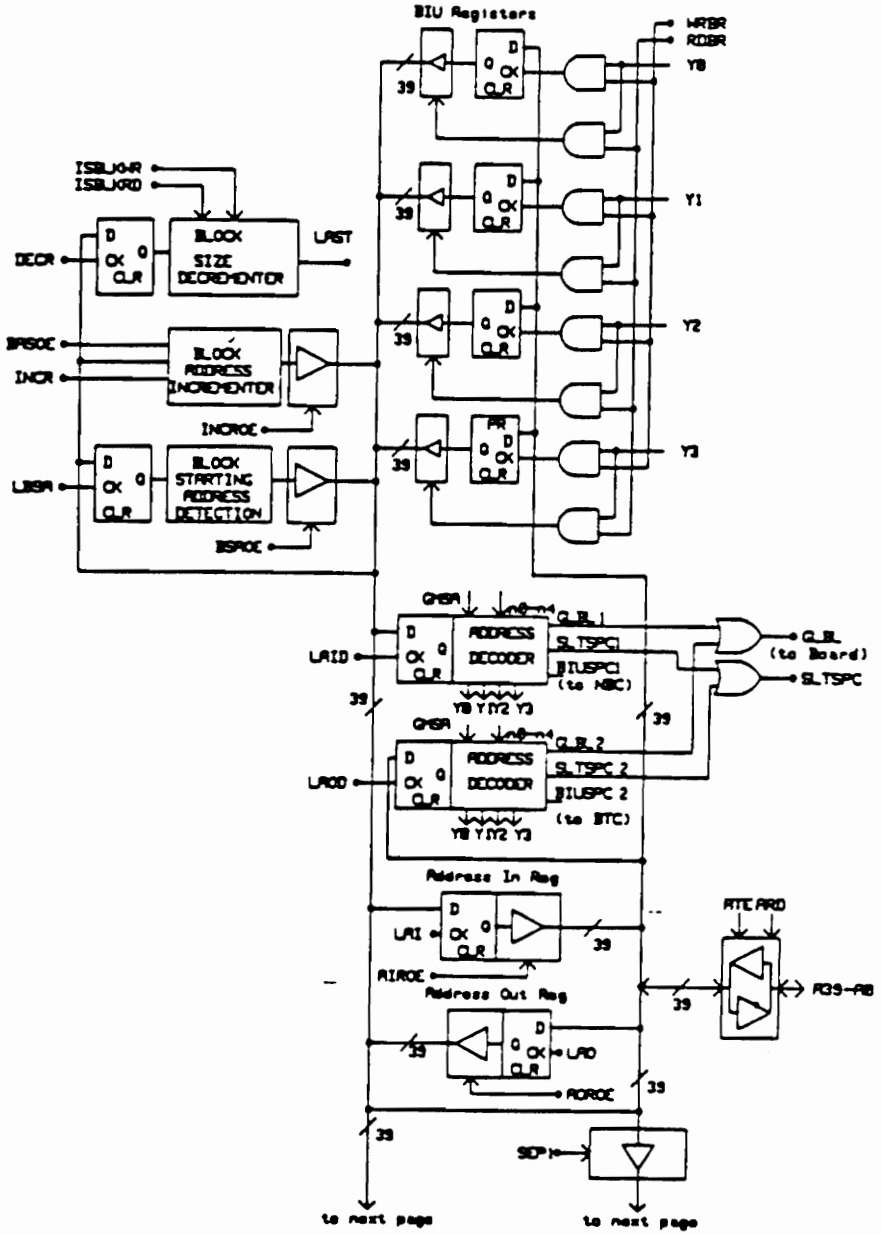


Figure 49. Block diagram of B-chip.

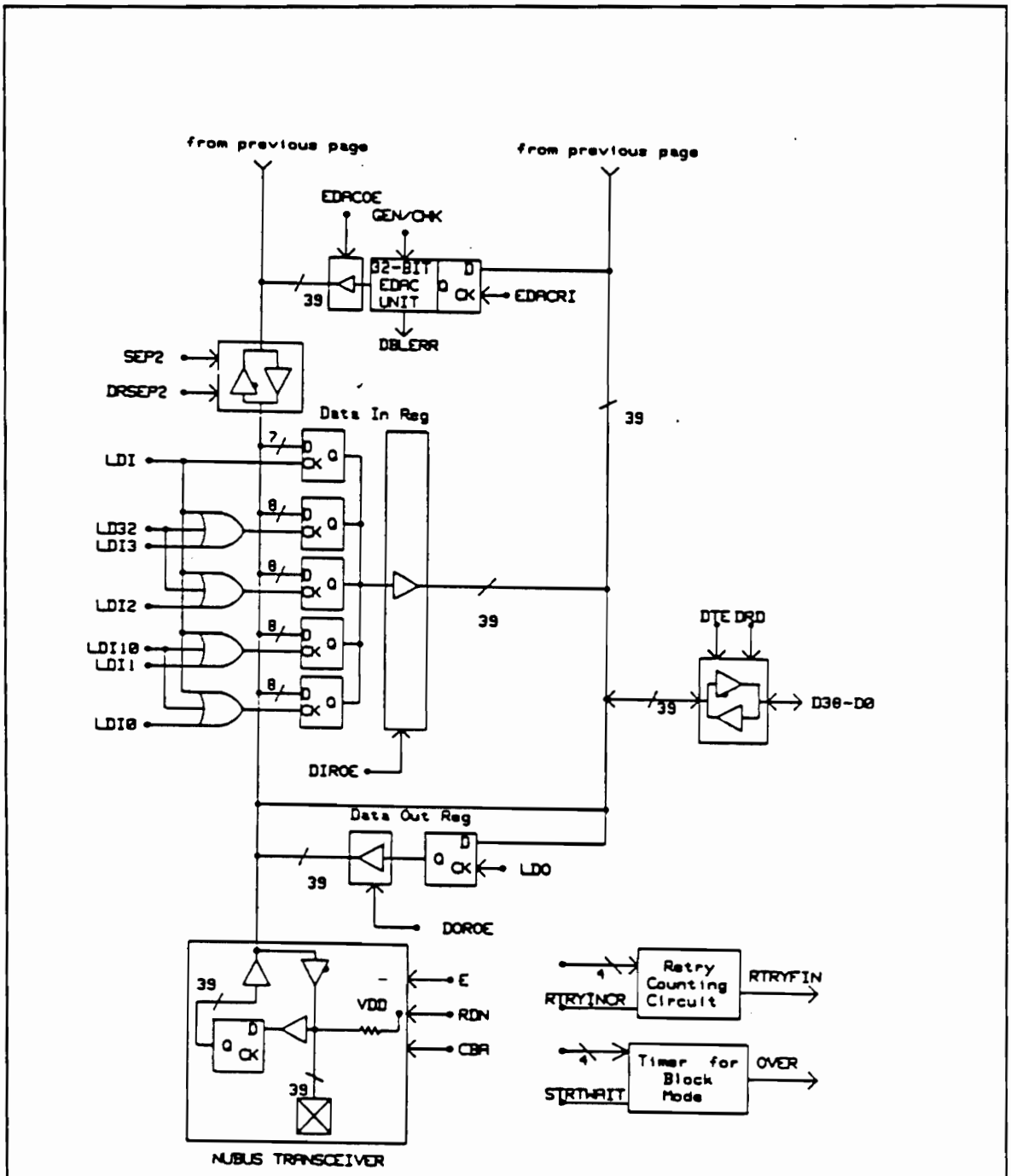


Figure 50. Block diagram of B-chip (continued).

6.2.2 Data Path

As can be seen from Figure 49 on page 117 and Figure 50 on page 118, the internal data path in the B-chip is multiplexed with the address path. Both the BTC and the NBC use the same data path to perform their respective operations. Since the address and data buses are demultiplexed on the board, two board transceivers are used and are controlled by the ATE, DTE (address and data transceiver enable), ARD, and DRD (address and data transceiver direction) signals. If a master mode write operation is to be carried out, both the address and data are present at the same time. Similarly, if it is a slave mode write operation, both address and data should be placed on the board-BIU interface at the same time. In such situations, the data word and address word will be present on the internal bus at the same time. Hence, tri-state isolators are required.

Two of the tri-state isolators have been used in this data path design. One is unidirectional and is controlled by the SEP1 signal, while the other one is bidirectional and is enabled and disabled by the signal SEP2 with its direction being controlled by the DRSEP2 signal. When the signals SEP1 and SEP2 are disabled (a low on each of them), the data path is functionally separated into two parts, and then the address and data can be present on the bus at the same time.

For example, in the case of a master mode write operation, both SEP1 and SEP2 are disabled, and the address and data from the board-BIU interface are latched into the Address Out Register and the Data Out Register by the LAO and LDO signals, respectively. To generate/check check bits on the address, SEP1 is enabled while SEP2 is disabled. By doing so, the Address Out Register is connected to the EDAC unit. The address is put on the internal bus by enabling AOROE and is latched into the EDAC unit by the signal EDACRI. The output of the EDAC unit is enabled using EDACOE and the modified word is stored back in the Address Out Register using signal LAO, afterwhich SEP1 is disabled.

Similarly, when generating/checking check bits for the data word, the SEP2 is enabled with SEP1 being disabled and DRSEP2 being low. The data word is put into the EDAC unit and latched back into the Data Out Register.

In the case of a slave mode operation, the slave BIU must latch the address on the NuFTbus whenever the NBC detects a START cycle. Since the address on the NuFTbus lasts for only one bus cycle (100 ns) and the NBC cannot determine whether or not the data path is available during this period. Therefore, a data latch is incorporated into the NuFTbus transceiver and the incoming address is stored in this latch by signal CBA which is issued from the NBC. Whenever the NBC has the control of the internal bus, the address is passed to the Address In Register by the signal LAI while enabling SEP2 and keeping DRSEP2 high.

6.2.3 Address Decode Unit

Since the BIU may be addressed as the master BIU and also as a slave BIU at the same time, there must be two address decode units to decode both the address coming from the board and the address coming from the NuFTbus to determine whether this BIU has to participate in that particular transaction. These two address decode units are functionally identical. One is associated with the BTC to decode the address from board and the other is associated with the NBC to decode the address from the NuFTbus.

The inputs to the address units are listed below:

- slot ID (4 bits with the parity bit being excluded)
- size of the on-board global memory (n: 5 bits in BIU register)
- start address of the on-board global memory (stored in BIU register)
- incoming address (either from board or NuFTbus)

The output signals from the address decode unit are listed below:

- **SLOT**: SLOT goes high to indicates that the incoming address is in the slotspace of the board.

- **GLOBAL**: GLOBAL goes high to indicate that the incoming address is the global space on the board.
- **BIU**: BIU goes high to indicate that the incoming address is one of the four BIU registers' addresses.
- Y_3-Y_0: Each of them goes high to select one of the four registers in the BIU.

If none of these signals goes high, then the address is in the slotspace of some other board or another part of global memory.

6.2.3.1 Address Detection Mechanism: The addresses of the BIU registers are assumed to be located at the first 4 words (first 16 bytes) in the slot space. So, AD_3 and AD_2 are used to select one of the four registers in the BIU. The size of the on-board global memory is given as a 5-bit binary number n. The number n indicates that there are 2^n words (2^{n+2} bytes) of global memory allocated to this slot and is loaded in the Board Status Register in the BIU during the initialization. The 5-bit number n cannot be greater than 30 since there are only 4 giga bytes in the NuFTbus memory space. The on-board global memory is contiguous. The 4-bit ID code is needed to determine the board's address in the slotspace. Slotspace for a board with an ID code of S_i is FS_iXXXXXXH. The ID code is wired into the card once it is inserted into the bus backplane.

So, each board has a slotspace ranging from FS_i000000H to FS_iFFFFFFH. Address lines A_{31}-A_{24} of the incoming address are compared with FS_iH. If they match, the output signal SLOT goes high in about 21 ns. The schematic is shown in Figure 52 on page 123. Global address is detected by masking the lower address. For the 5-bit memory size value n, the lower n + 2 address bits are masked (set to 0) and then this modified incoming address is compared with the start address, If they match, the output signal GLOBAL goes high in about 15 ns. This is shown in Figure 51 on page 122. The actual circuit for address decoding operates as follows. A macro cell PUT532 which is designed using LOGEN is used to generate the mask. Using the 5-bit input n, the masking pattern X_{31}-X_2 is generated, where X_{31}-X_{n+2} are ones and X_{n+1}-X_2 are zeroes. The incoming address is ANDed with the masking pattern to produce an **effective** starting address. Then

the address is compared with the global memory starting address coming from the Global Memory Starting Address Register in the BIU.

6.2.3.2 Simulation Example and Result: The address decoding circuit is simulated as given in the following example and the simulation results are shown in Figure 53 on page 124. Y_3-Y_0 are the decoded from signals BIU, IA2 and IA3 to select the registers in the BIU. Those glitches shown in the timing diagram are an inherent feature of combinational logic circuits because there may be more than one paths with different delays that determine the state of the output signals. Consider the SLOT signal, let the ID code for a board be 0110. Then its slotspace is F6XXXXXXH. Address signals A_{31}-A_{24} of the incoming address are compared with F6H. Thus, any address between F6000000H and F6FFFFFFH is detected as the address in the slotspace, and, correspondingly, the output SLOT signal goes high.

Consider the GLOBAL signal, let n for this board be 01110. Then the on-board global memory has 2^{14} words (2^{16} bytes). Let the start address of the on-board global memory be BA230000H (Note that the last n + 2 bits of the start address are always zeroes). Hence, the global memory range is from BA230000H to BA23FFFFH. Let an incoming address be C12B80A6H. The lower n + 2 address bits are masked by the address decode logic. So, C12B0000H is compared with BA230000H. Since they do not match, the incoming address is not in the global address space of the board. Now, let the incoming address be BA232345H. After masking off the lower 16 bits, incoming address matches the start address. So, the output GLOBAL signal goes high to indicate that the address is in the global memory space on the board.

Now consider the BIU and the Y_3-Y_0 signals. Once the SLOT is already high and address lines A_{23}-A_4 are all zeroes, (since the four registers occupy the first four words in the BIU memory space) the BIU signal goes high and Y_3-Y_0 go high according to the states in the A_3 and A_2.

6.2.3.3 Relationship Between Address Decoders and Board: The BTC and the NBC interpret the outputs from the address decode unit in different ways. For an address generated by the NuFTbus,

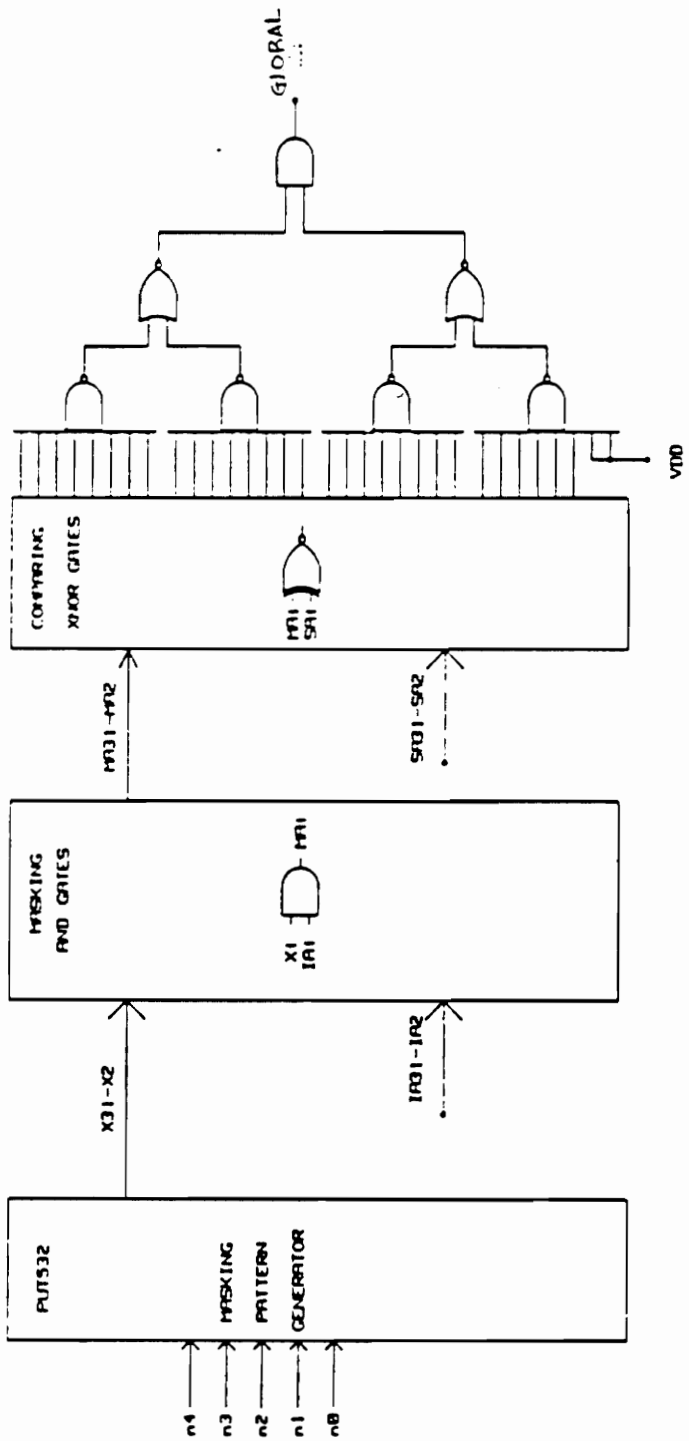


Figure 51. Address decode unit.

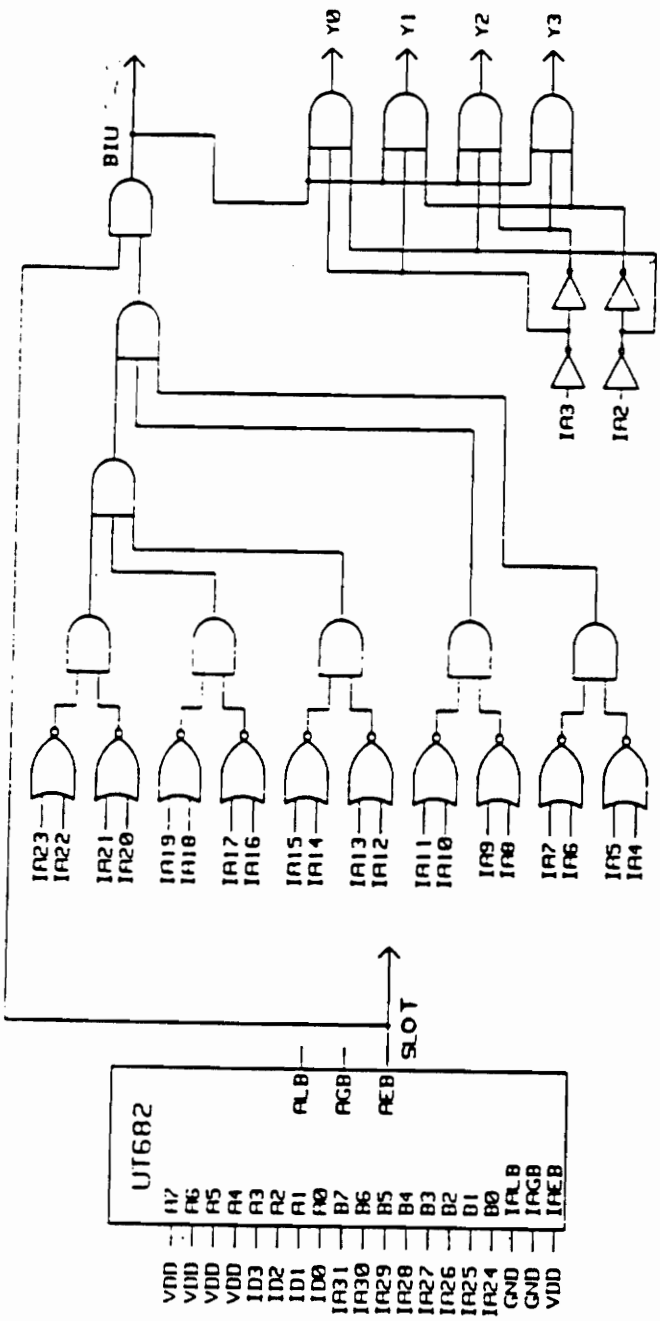


Figure 52. Address decode unit (continued).

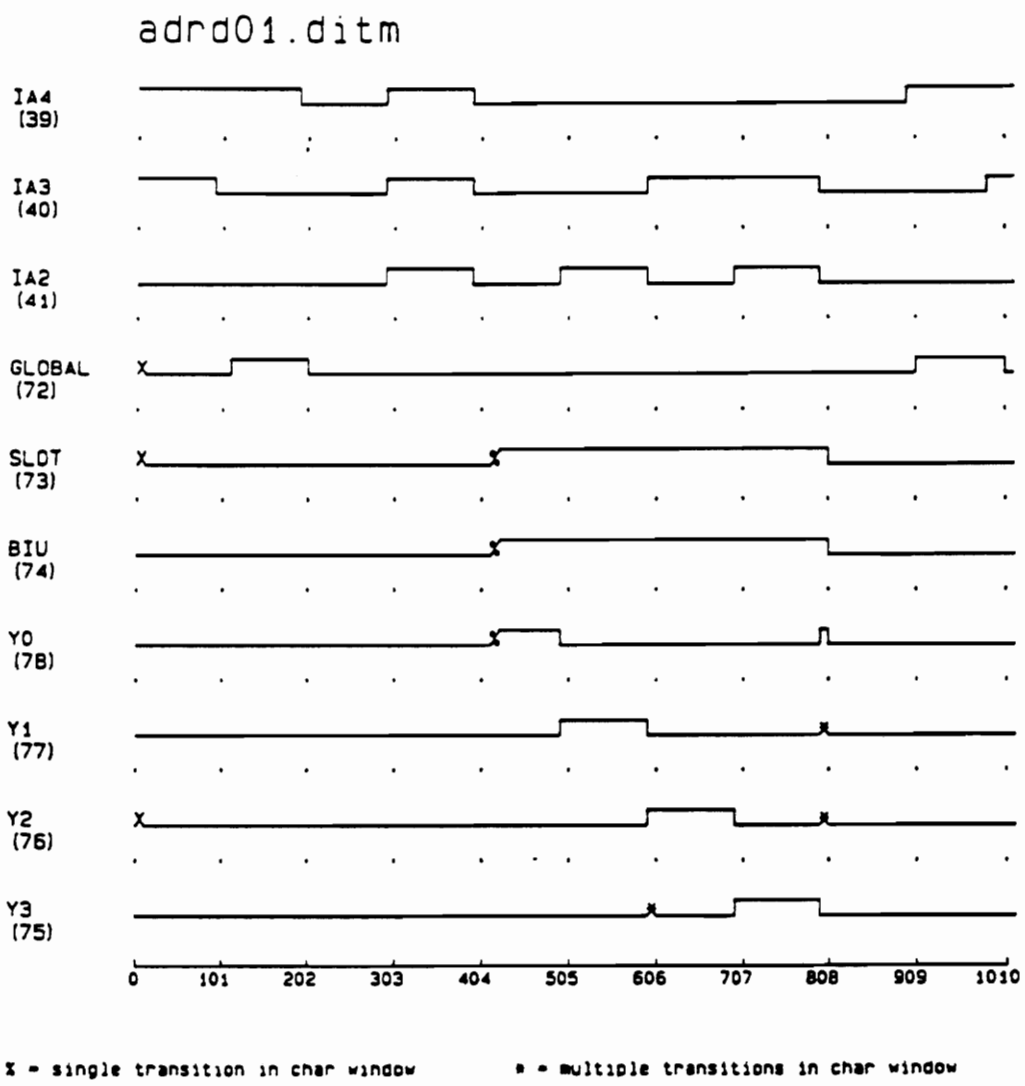


Figure 53. Simulation result for the address decoder.

the outputs of the address decode logic tell the NBC whether there is to be a slave operation on the board or on the BIU itself. If so, then the NBC starts a corresponding slave read/write transaction. For an address generated by the board, the outputs of the address decode unit going high tells the BTC that the operation is to be completed on the board itself. In such a case, the BTC will take part in the operation only when BIU is high, otherwise, it will do nothing but monitor the SENI line util the operation is completed.

The SLOT signals from each of the two address decode units are ORed together and passed to the board. Similarly the two global signals are ORed together and passed to the board. This is useful in two ways. First, the board does not have to have the same decoding logic which is already present in the BIU. The board does the decoding only of the lower address lines. Secondly, in the case of an address from the NuFTbus, the memory on the board gets advanced notice of the forthcoming memory related operation before the NBC and the BTC finish other tasks and pass the address to the board.

6.2.4 BIU Registers

As stated in chapter 5, there are eight registers in the BIU. They are:

- **Address In Register**: Used to store the address coming from the NuFTbus and is controlled by the LAI (latch address in) and AIROE (address in register output enable) signals.

- **Address Out Register**: Used to store the address going to the NuFTbus and is controlled by the LAO and AOROE signals.

- **Data In Register**: Used to store the data coming from the NuFTbus and is controlled by the LAI and DIROE signals.

- **Data Out Register**: Used to store the data going to the NuFTbus and controlled by the LDO and DOROE signals.

The following are the special registers storing the control information and status from bus transactions:

- **Global Memory Starting Address Register**: This register stores the start address of the on-board global memory and is selected by the signal Y_0 from address decode unit and controlled by the RDBR (read BIU register) and WRBR (write BIU register) signals.

- **Board Status Register**: This register stores the size of the global memory on the board, and is selected by the signal Y_1 and controlled by the RDBR and WRBR signals.

- **BIU Command Register**: This register stores the number of retries for failed transactions, maximum number of transactions allowed in the lock mode of operation, and the number of clock cycles to wait for the slave to proceed with the next data transfer in the block mode operation. It is selected by the signal Y_2 and controlled by the RDBR and WRBR signals.

- **BIU Status Register**: This register stores the following flags TMOUT (time out acknowledgement from the slave), RTRYFAIL (all retries failed), BIUMERR (double error in the BIU register), ADDRERR (double error in the address word), MEMERR (double error in the data word), and ARBFAIL (error in arbitration). It is selected by the signal Y_3 and is controlled by the RDBR and WRBR signals.

A schematic diagram of the circuit used for performing read/write operations on BIU registers can be found in Figure 49 on page 117.

6.2.5 32-bit EDAC Unit

The Error Detection and Correction (EDAC) unit contains the logic necessary to generate the check bits on the input data word according to a modified Hamming Code, and to correct the data word when check bits are supplied. Operating on the input data, the EDAC unit will correct any single bit error (both data word and check bits), and will detect all double bit errors. For a 4-bit word, 4 check bits are used while for 32-bit words, 7 check bits are used [21].

There are two EDAC units in the BIU. One is the 4-bit EDAC unit used in conjunction with the control lines in the A-chip, and the other is the 32-bit EDAC unit used with the address/data word. The 4-bit EDAC unit is used exclusively by the NBC while the 32-bit EDAC unit is shared by both the BTC and the NBC as described in section 6.0.5.

The following is the list of the input and output signals of the EDAC unit&gm.

- **EDACRI** (input): A rising edge on EDACRI latches the data word and the check bits into the EDAC unit.

- **GEN/CHK** (input): When GEN/CHK is high, the EDAC unit is in the generating mode with AEPR/DEPR from the board being low. The EDAC unit generates the check bits for the incoming data. When GEN/CHK is low (AEPR/DEPR is high), the EDAC unit is in the checking mode. It checks for single and double bit errors and will correct the single bit error in both the data word and check bits.

- **EDACOE** (input): A high on EDACOE enables the register output. The outputs are tristated when EDACOE is low.

- **DBLERR** (output): DBLERR going high indicates that there are two or more than two bit errors in the data word and check bits.

The 32-bit EDAC unit is designed using LOGEN and its block diagram is shown in Figure 54 on page 129.

6.2.6 Block Starting Address And Block Size Detection

When operating in the block mode, the size of the block to be transferred and bits A_5-A_3 of the starting address are determined by an encoding of the AD_5-AD_2 lines [11]. The schematic for block starting address detection of the address bits A_5-A_3 is shown in Figure 55 on page 130. The schematic for block size detection is shown in Figure 57 on page 133. **Note**: in the block size detection, the block size decoding is designed to be one word less than the actual block size. So, the possible block sizes are 1, 3, 7, and 15, corresponding to 2, 4, 8, and 16, respectively. This is for the convenience in the block size decrementing circuit design.

6.2.7 Block Operation Address Incrementer

In a block mode operation, only the start of block address is sent on the NuFTbus from the master; no intermediate addresses are sent by the master. It is the slave module's responsibility to provide the sequential ascending addresses and to receive the data words in a block write operation, or to send the sequential data words in a block read operation. The increment operation is performed by the NBC in the BIU. The schematic for address incrementing is shown in Figure 56 on page 132 and is cascaded to make a 32-bit incrementer with A_2-A_0 always being 000.

Whenever there is a block mode slave operation, the NBC latches in the address, checks the validity of the address word using the EDAC unit, and then sends this address to the block starting address detection module. After these tasks are finished, the NBC asserts the signal BSAOE (block starting address output enable) and puts this address on the board-BIU interface. At the same time, with BSAOE being high, the NBC asserts the signal INCR to load the address into the address

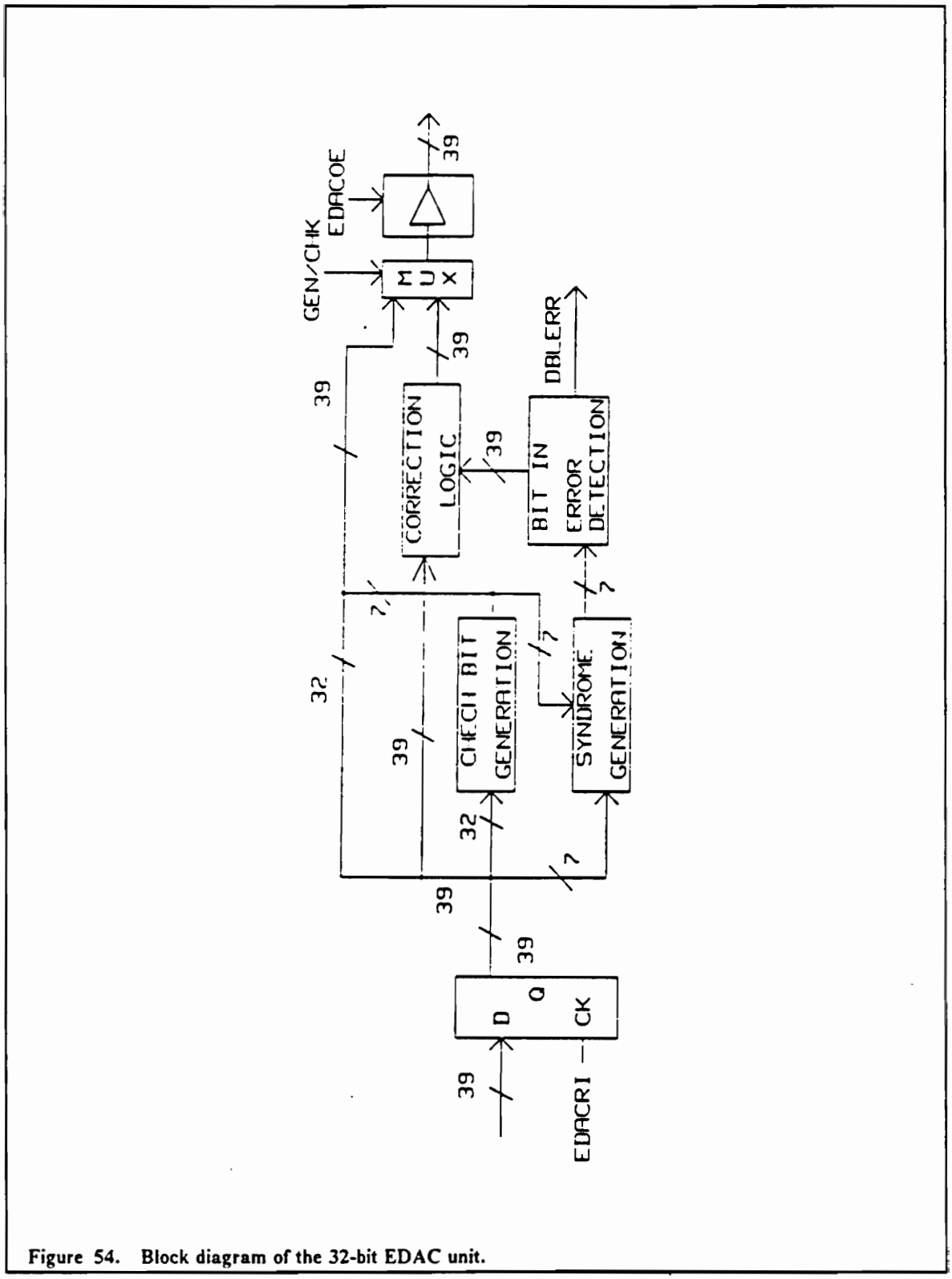


Figure 54. Block diagram of the 32-bit EDAC unit.

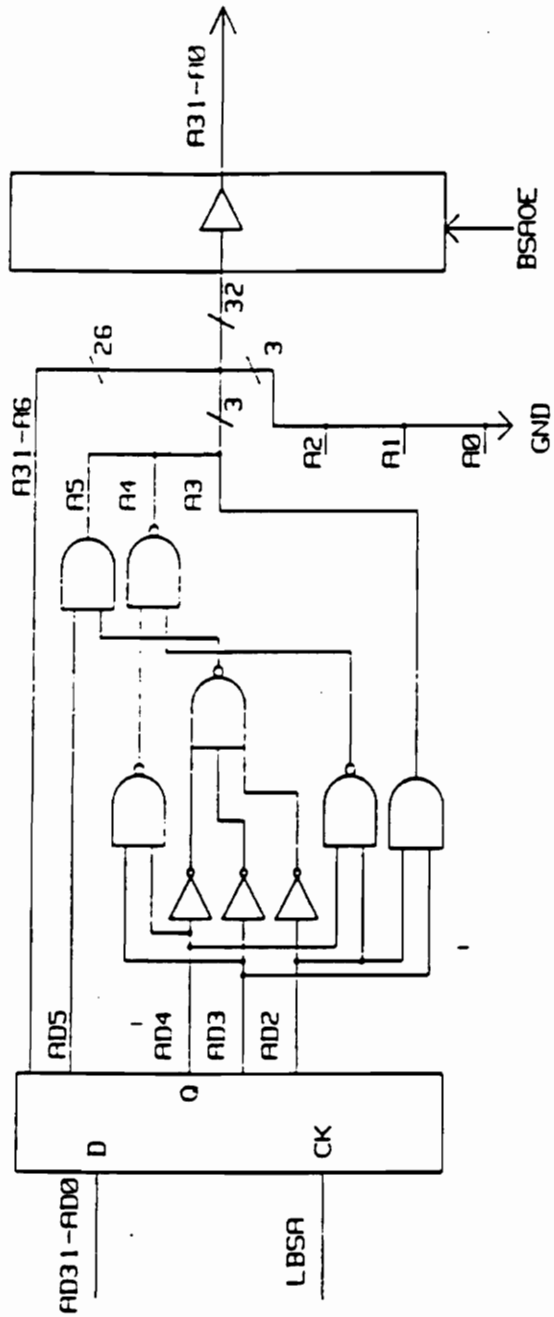


Figure 55. Block starting address detection.

incrementer without increment on the address (since BASOE is high and C_{in} is low, and a low on C_{in} will disable the circuit from incrementing). Since the signal BSAOE is high only this time during the whole NuFTbus block mode operation (because the block starting address detection is performed only once for each transaction), therefore after the first operation, the address provided to the incrementer is from the incrementer itself as shown in the circuit. After the BASOE is set to low, the address is incremented by a rising edge on the signal INCR and is put on the bus by enabling the signal INCROE.

6.2.8 Block Size Decrementer

This circuit is shown in Figure 57 on page 133. The block size is decremented each time an intermediate data cycle is completed. Whenever the signal LAST goes high, it indicates that there is only one data cycle left unfinished in this block mode operation. Thus, the NBC can determine when to terminate the block operation.

6.2.9 Timer to Wait Between Transfer in Block Mode

In block mode operation, the NuFTbus does not provide a signal which would indicates that the intermediate data transfer is completed and the next transfer should be started. If some kind of synchronization mechanism is not provided during the block mode operation, the data might be lost/corrupted and the master BIU might wait indefinitely for the data. This is explained as follows.

In the case of a block read operation, the slave BIU puts the data on the NuFTbus along with an intermediate acknowledgement. The master fetches the data and sends it to the board. After this is done, the master BIU just waits for the next data from the slave module. In this case, if the master board is slow while the slave BIU does not wait a certain period of time, the master BIU might miss the next data from the slave BIU and wait indefinitely.

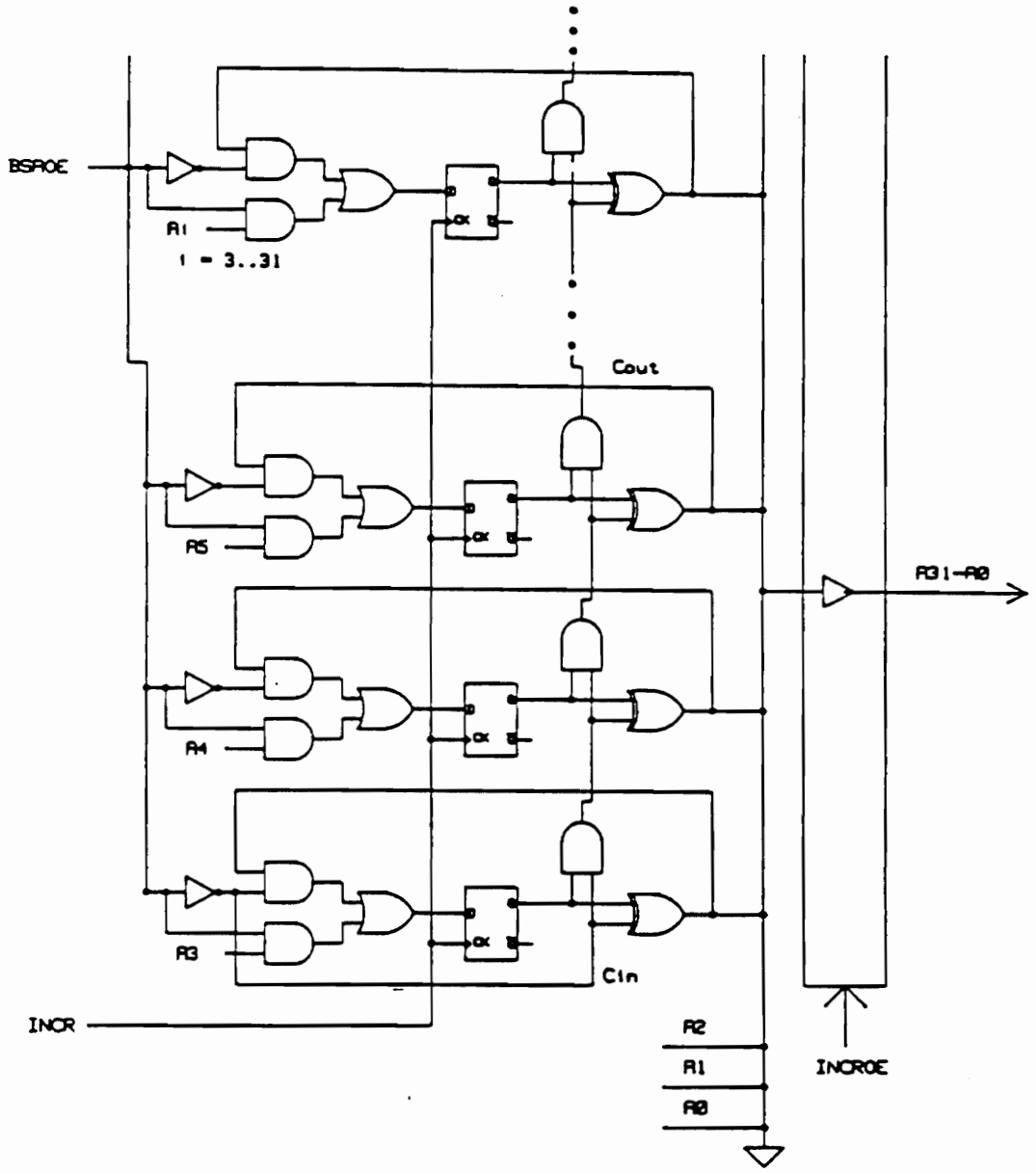


Figure 56. Block address incrementer.

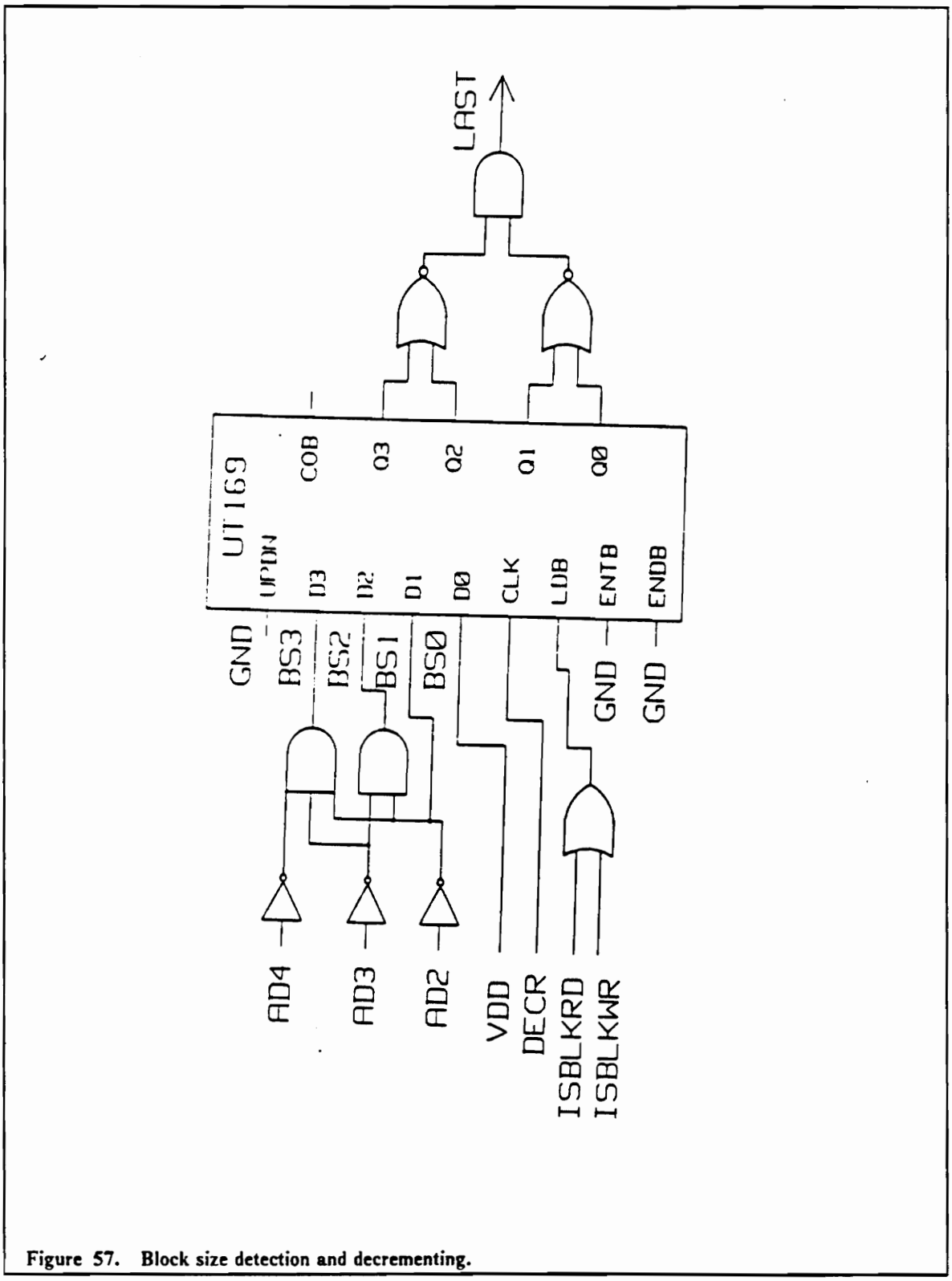


Figure 57. Block size detection and decrementing.

To get around this problem, the slave NBC waits for a certain period of time before it starts the next data transfer. An 8-bit register in the BIU is used to store the number of clock periods that the NBC should wait before starting next transfer. During initialization, a number is loaded into this register to indicate the clock cycles to wait between two intermediate data transfers. The circuit to implement this timer is shown in Figure 58 on page 135. Once the slave NBC finishes a data transfer, it gives out a STRTWAIT signals and starts counting. When the count matches the number coming from the BIU register, a signal OVER is generated and tells the NBC that it can proceed with the next data transfer. A high on signal OVER also resets the counter.

6.2.10 Retry Counting Circuit

If a NuFTbus transaction fails, the master BIU retries it before it informs the board about it. The number of retries is preloaded in the BIU register at the time of initialization. When the master BIU retries its operation, it increments the count. When the number of retries is reached, the master NBC get a RTRYFIN signal from the counter. Then it sets a flag in the BIU register and gives out an error signal to the board. This retry counting circuit is shown in Figure 59 on page 137.

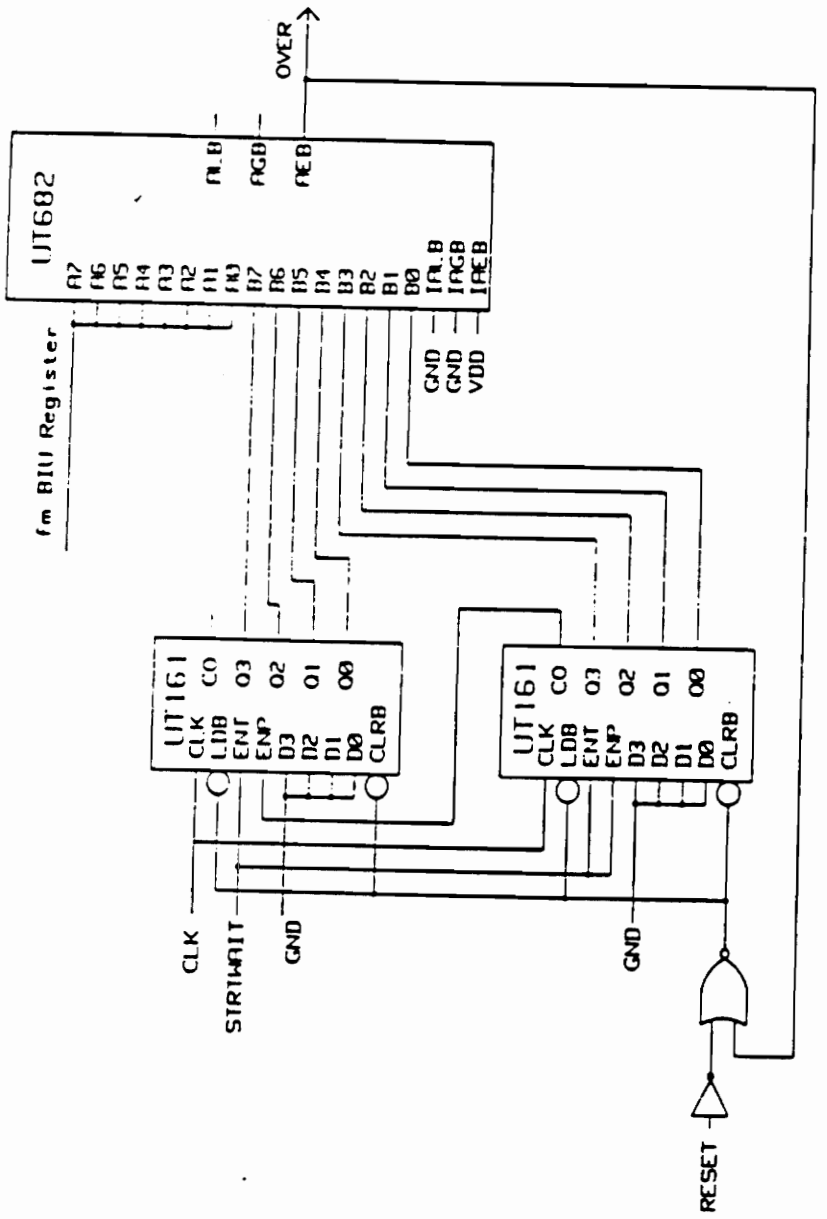


Figure 58. Timer to wait between intermediate transfers.

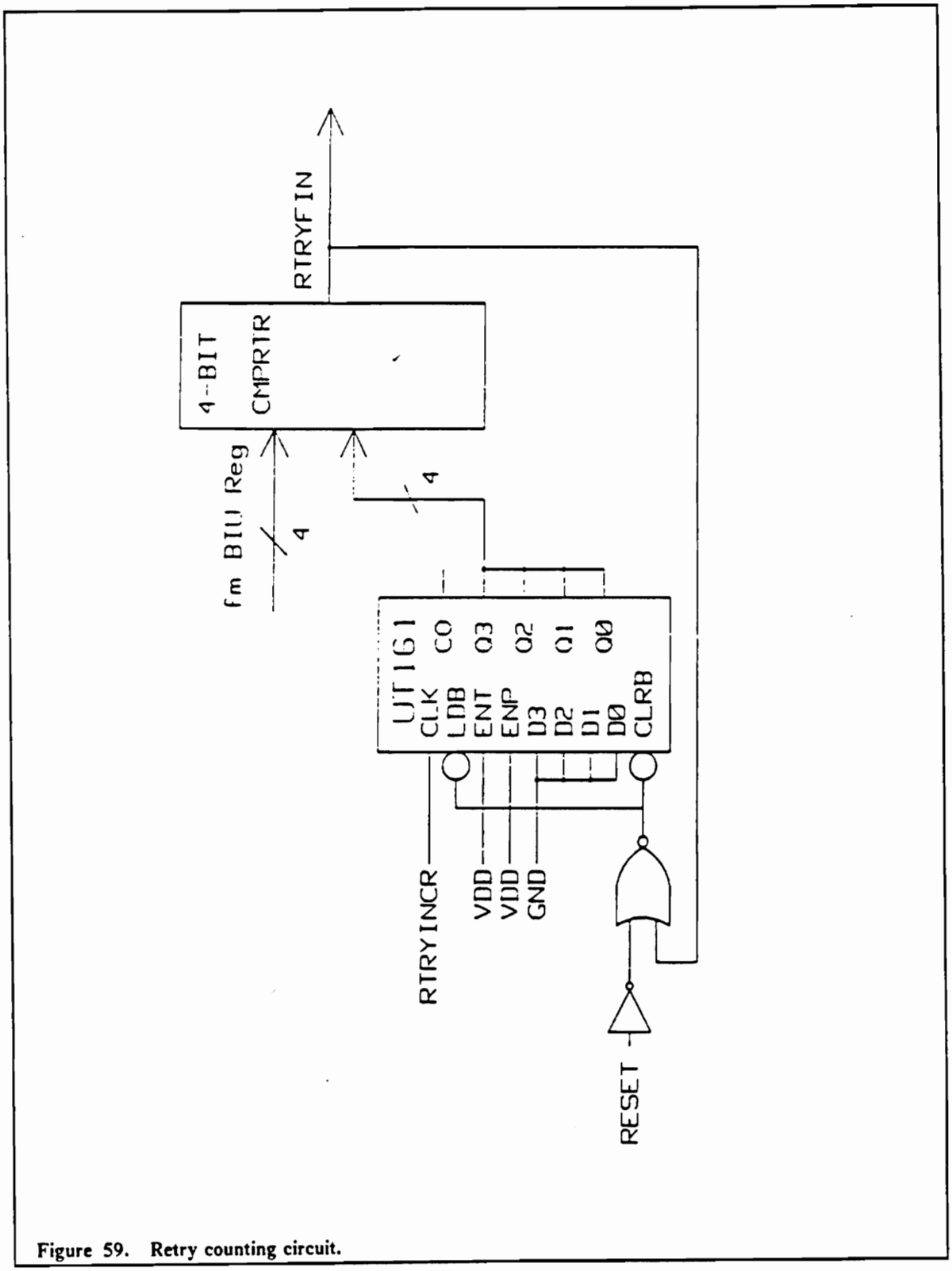


Figure 59. Retry counting circuit.

7. Problems Encountered and Possible Future Work

This chapter explains the problems encountered during the BIU design, outlines possible future work and thereby provides a conclusion to this thesis.

7.1 Complexity, Speed and Structure

According to the NuFTbus specifications, the central bus clock frequency is 10 MHz. So, the internal clock frequency for the finite state machine must be fast enough to finish all the operations in time to latch in information from the NuFTbus or send out information into the NuFTbus. But for gate array ICs, with other variables such as voltage and temperature constant, the speed depends on the following factors:

1. Current IC technologies: in general, an IC fabricated with a bipolar technology (usually it is ECL) is faster than those built with CMOS technology.

2. Current process technologies: this will determine the smallest size for the transistors that can be manufactured with a reasonable yield rate. Smaller channel length transistors will have faster switching speeds.

3. The complexity of the circuit itself: the bigger the circuit is, the more propagation delay. Furthermore, this propagation delay in the combinational logic of a finite state machine will affect how fast the clock for this finite state machine can run.

4. The length of the routing path between gate array macro cells: this depends on the routing algorithm and how well circuit modules are placed.

In the NuFTbus BIU design, the EDAC unit and the control circuits (BTC and NBC) are the speed-sensitive modules and make the BIU too slow to meet the NuFTbus specifications. This will be explained below.

The gate counts in each state machine of the BTC and NBC are very large; the biggest one has about six hundreds gates in its combinational logic. Therefore, the long time delay in the combinational logic makes the finite state machines slow. Besides, the BTC and NBC are simulated under condition as high temperature low voltage, as stated in the beginning of chapter 6, which makes the circuits even slower.

As for the 32-bit EDAC unit, by using UTMC's UTDR cell library (United Technologies's radiation hardened gate array cells) [23] and simulating under CONDITION D (high temperature and low voltage), the propagation delay to detect a double bit error is 29 ns (from when the EDAC unit latches in the data/address word to when double bit error flag is up), while the time to correct single bit error is 53 ns (from when the EDAC unit latches in the data/address word to when the corrected data/address word appear on the output of the EDAC unit). These timing data are derived from the simulation output report file. This is slow when compared with the standard EDAC ICs on the market because the vendors can use all the technologies that are available in full custom design, such as sacrificing area in the critical path for speed improvement, using pass-gate or transmission-gate innovative design, and smaller transistors (one micron instead of 1.5 micron provided by UTMC) [21,24]. For instance, AMD's Am29C660C, when operating over COMMERCIAL ranges, takes 16 ns to flag up a double bit error and 24 ns to correct a single bit error.

Unless this BIU will be designed and manufactured using full-custom technologies to optimize the speed of the EDAC unit, the delay caused by the EDAC unit is too long to be negligible when system performance is concerned. Certainly, no matter how fast the EDAC unit is, it will degrade the system performance to some extent.

With current gate array technologies from UTMC, the following are possible directions worthy of future efforts.

1. When a finite state machine is designed using the PEG compiler and an output signal must be held high across several states, this output signal must be asserted in all the states in which it is intended to be high. As this kind of output signals is concerned. we can use SR flip-flops to keep track of status. For example, if a signal SIG must be held high from state 0 to state 3 of a certain finite state machine, a signal SSIG which is connected to the S input of the SR flip-flop can be asserted in state 0 and a signal RSIG which is connected to the R input of the SR flip-flop can be asserted in other state that is transited from state 3, and SIG is the Q output of the SR flip-flop. This may increase the number of output signals of the finite state machine but will not necessary increase the total hardware since there are less output signals appearing in each state.

2. From the McLDL files for the finite state machines BTC and NBC, we can see that most of the circuitry is used to control the state transitions. Hence, to speedup the finite state machines, one approach is to further partition them into even smaller machines with fewer states in each, and thus reduce the logic in state transitions.

3. To save circuitry in the BIU, the BTC and the NBC, the 32-bit EDAC unit is shared between these circuits. Much time is spent in handshaking. One possible configuration that can speedup the operation is to use two 32-bit EDAC units and save the handshaking time. To proceed one step further, one could use four EDAC units in the BIU design. With two EDAC units for each controller, the EDAC operation on address and data (if a write operation occurs)

can be carried out in parallel rather than in series as in the current design. Besides, these EDAC units can replace the Address In Register, Address Out Register, Data In Register, Data Out Register in Figure 42 and Figure 43 and therefore can save the time needed to transport back and forth the address and data between the EDAC unit and the appropriate registers.

7.2 Testable Design For Gate Array ICs

Design for testability involves two major attributes: controllability and observability [25]. To obtain controllability, one must be able to control the logic state of a particular node in order to activate the faults on that node. To activate a fault means to set the logic value of a node to the opposite state of the stuck-at value being tested for. For example, testing a node for a stuck-at-0 condition requires controlling that node to a logic-1 value. Observability can be thought of as the difficulty encountered when attempting to propagate the effect of the fault activation to a primary output for fault detection. Propagation requires that the response of the network to the activated fault be observed from at least one primary output. These two properties are not mutually exclusive. Circuits with a high degree both of controllability and observability make the test generation process less difficult.

A testable design contains the following general properties [26].

1. It contains no logical redundancy which will make a stuck value on a line undetectable.

2. It contains no asynchronous logic. Glitches and transient pulses coming from combinational circuits, when appearing at the input of the unclocked flip-flops, may make them store the incorrect value. A test generator for this kind of circuits is much more complicated since it has to deal with circuit delays in order to create hazard-free tests.

3. Its clocks are isolated from the logic. This enables the tester to control the unit under test at the speed of the tester rather than the speed of the unit.

4. Its sequential circuits are easy to initialize. Many test vectors can be saved when those elements can be initialized either at power-up or by certain external controls.

5. It is easily diagnosable. It avoids ambiguous fault isolation situations such as high fan-in wired AND or OR circuits.

6. It has a minimal increase in gate or pin count over a normal design.

Several different techniques for testable design have been proposed and widely used [25]. These techniques can be broken into two categories: ad hoc and structured. Ad hoc approaches include partitioning, degating, adding extra test points, bus architecture systems, and signature analysis. By contrast, structured approaches have the objective of being able to observe and control the state variables of a sequential machine. There are four categories of structured approaches: random access scan (a multiplexer technique), level-sensitive scan design (LSSD) and scan path, scan/set logic, and built-in logic block observation (BILBO) [17].

7.2.1 Proposed Testable Design For The BIU

There are two chips in this version of the NuFTbus BIU: A-chip and B-chip. Essentially, the A-chip is a data path chip and contains no sequential circuits. Since all the control signals are from the B-chip and all internal registers can be made accessible at the address and data buses, no special testability features are needed in the A-chip. As for the B-chip, the proposed testable designs are presented in the following sections for the arbitration unit and the controllers, BTC and NBC.

7.2.2 Arbitration Unit

Redundancy has been added to the arbitration unit to increase its fault tolerance, therefore test points are a crucial addition to ensure that this device is operating correctly without any portion in it being faulty. The configuration of the arbitration unit with test points inserted is shown in Figure 60 on page 144. Without the these added test points, it is entirely possible that half of the redundant circuitry could be faulty, and the test for manufacturing faults being executed could still pass. Adding test points at the output of each duplicated portion of the redundant circuit allows for independent testing of each section. However, these test points must be made observable in the I/O pins by the scan-set technique proposed in the next section.

7.2.3 BTC And NBC

Scan-set techniques can be used to control and observe those signals that do not appear in the I/O pins of the gate array ICs. As shown in Figure 61 on page 145, the inputs of an N-bit wide serial shift-register are connected to various internal signal nodes. Under control of a load/shift control line, a snapshot of the logic states of the output signals of the state machines can be loaded in parallel into the shift-register, then shifted out, by applying a scan clock. Using some extra gating logic and by shifting a pattern into the register, the states of internal nodes can be set from the primary inputs.

7.3 Conclusion

The NuFTbus bus interface unit is to be used in a parallel system bus in a multi-processor system. The system is to be used as a part of a satellite system. In order to provide dependable operation, the bus interface unit must be fault tolerant.

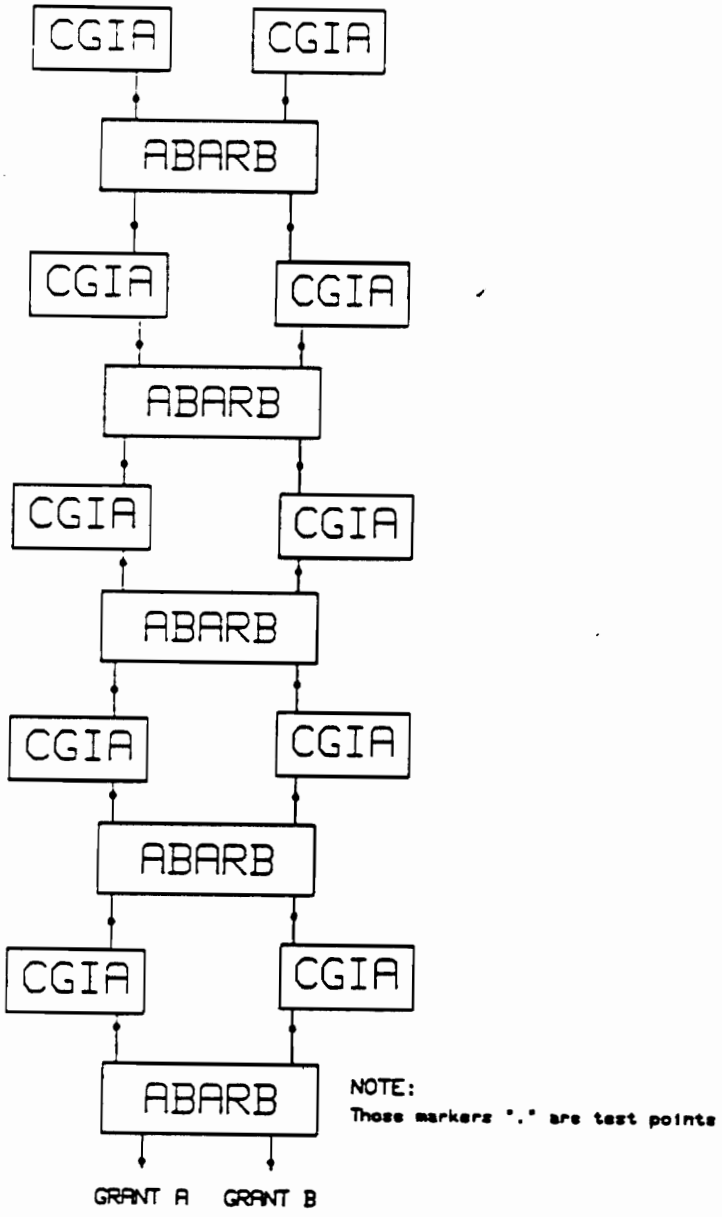


Figure 60. Testable feature added to the arbitration unit.

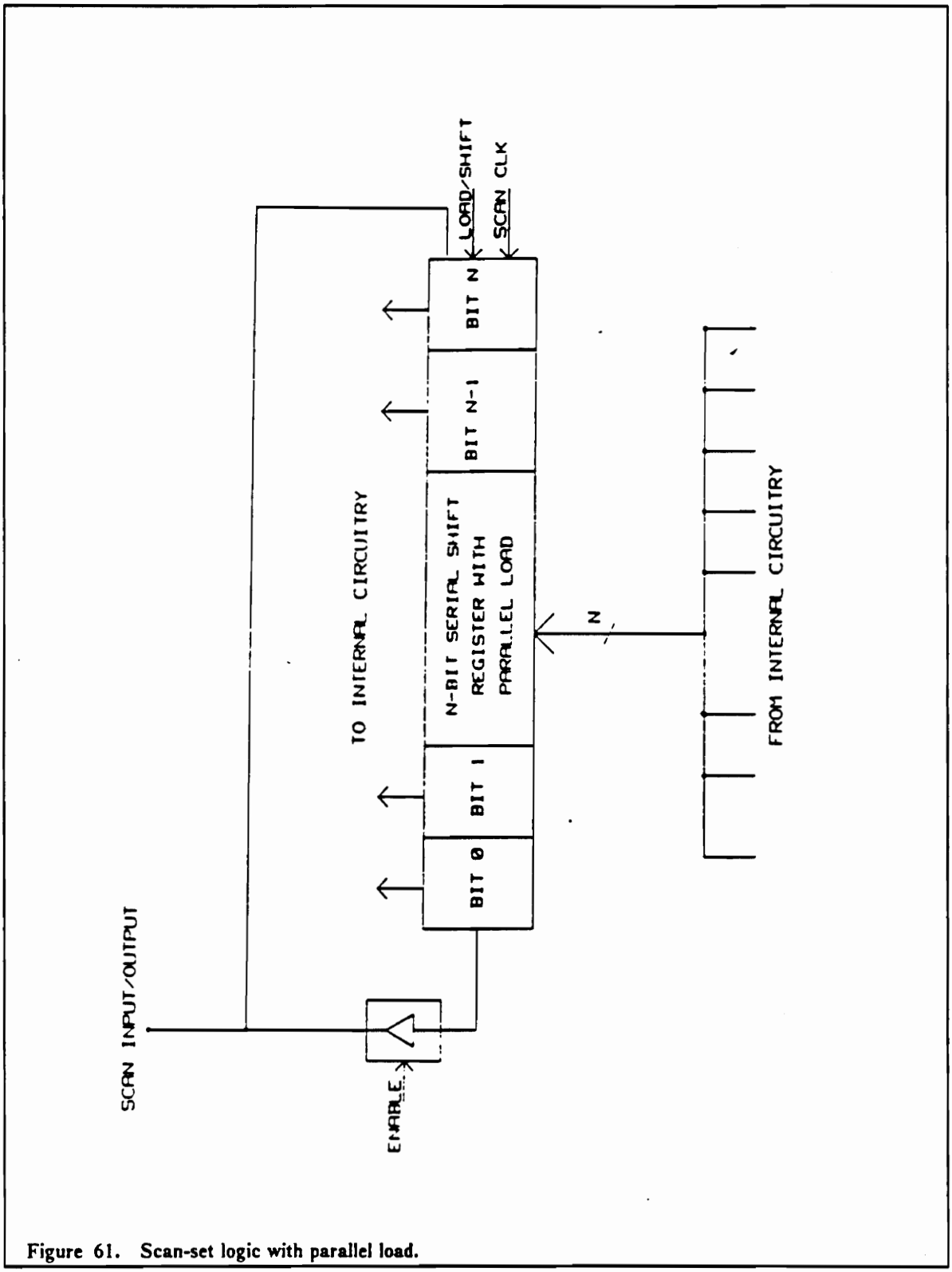


Figure 61. Scan-set logic with parallel load.

The NuBus architecture was selected to implement the fault tolerant bus NuFTbus because of its simplicity and low cost when compared with other buses in the same class of performance. The arbitration unit was duplicated in each BIU to provide for redundant fault tolerance since arbitration among different BIUs for bus ownership is an important criteria for the proper functioning of the system. Also, to maintain the integrity of the flow of control and data information, error detection and correction capability was incorporated into the BIU. This ensured that every word coming in or going out of the BIU was free from single bit errors.

Several CAD tools were used in the logic design of the BIU. Each circuit module was described by McLDL logic language and simulated with stimuli written in SIL simulation input language on the HIGHLAND Design System. The functionality of the controllers, BTC and NBC was simulated and verified according to different NuFTbus transactions such as single word transfers and block transfer.

The A-chip of the BIU needs about 18,000 transistor pairs to implement while the B-chip needs about 10,000 transistor pairs. The HIGHLAND system can support a design with at most 25,000 transistor pairs in one gate array.

7.4 Future work

The following steps should be taken in order to fabricate the BIU as a gate array IC.

1. Partition the state machines into smaller ones to gain speed to meet the NuFTbus specification. If this step can reduce the total logic in the controllers, an effort should be made to put all the circuitry of the BIU into a single gate array IC.

2. Refine the control sequence in the controllers to exploit parallel operations as much as possible. Consider using more EDAC units as stated in Section 7.1 or to leave the EDACs out of the BIU.

3. Before piecing all the circuit modules together, a complete testable design scheme should be imposed on the BIU as a whole.

4. A complete set of test vectors should be designed carefully to meet the fault coverage (greater than 90 percent) required by HIGHLAND.

5. After layout, if a ZYCAD Accelerator is available, run a multi-chip simulation to verify the function and timing of the BIU both as a master and as a slave.

Bibliography

1. B. Nicholson, "Synchronous 32-bit Backplane Buses Open Up Distributed System Design," *EDN,* June 14, 1984
2. S. Ohr, "Buscon: Busing is the hitch in multiprocessing's star," *Electronic Design,* January 22, 1987, Vol 35, pp.36-38
3. G. Laws, "Multiprocessing on the NuBus using Cache inhibited Pages," *IEEE Microprocessors and Microsystems,* April, 1988, Vol 121, No. 3, pp.147-152.
4. M. Wright, "Bus-interface ICs," *EDN,* May 26, 1988.
5. B. Johnson, "Design and Analysis of Fault-tolerant Digital Systems," Addison-Wesley Publishing Company, Inc., 1989
6. J. C. Pickel, J. T. Blandford, "Cosmic ray induced errors in MOS memory cell," *IEEE Trans. Nuclear Science,* Vol NS-25, No. 6, December 1978, pp.1160-1170.
7. J. P. Wood, D. K. Nichols, "Investigation for single event upsets in MSI devices," *IEEE Trans. Nuclear Science,* Vol NS-28, No. 6, December 1978, pp4022-4025.
8. R. Wolfe, "Impact of PCs on Automated Test Measurement," *Intelligent Instruments and Computers,* May, 1988, pp148-154.
9. G. Laws, "Introduction to the NuBus," Eurobus 89 Proceeding, May 9, 1989.

10. W. Nowlin, "PC Bus Performance - NuBus vs. MicroChannel," *Electronics & Wireless World (United Kingdom),* 1988, pp856-859.
11. IEEE Specification 1196, "NuBus - A simple Backplane Bus," 1987.
12. W. Nowlin, "Is the NuBus really a better bus ?", *Electronic Engineering Times,* February 6, 1989, pp.56,73.
13. G. White, "NuBus - A Simple, Cost-Effective Bus Interface," National Instruments, 1989.
14. P. Paranjape, "A fault-tolerant bus interface unit based on the NuBus standardized bus architecture," Master's thesis, Virginia Polytechnic Institute and State University, January, 1988.
15. M. Ercegovac, T. Lang, "Digital Systems and Hardware/Firmware Algorithms," John Wiley & Son, Inc., 1985.
16. S. Hurst, "Custom-Specific Integrated Circuits," Marcel Dekker, Inc., 1985.
17. E. Hollins, "Design of VLSI Gate Array ICs," Prentice-Hall International, 1987.
18. "HIGHLAND Reference Manual," United Technologies Microelectronic Center, Publication 94002.
19. "1986 VLSI Tools: Still more work by original artists," Report # UCB/CSD 86/272, Computer Science Division, University of California, Berkeley, December, 1985.
20. "LOGEN Tutorial, "United Technologies Microelectronics Center, May, 1987.
21. "High-speed CMOS Logic Data Book,"Texas Instruments, 1984.
22. R. Hamming, "Coding and Information Theory," Prentice-Hall, Inc., 1980.
23. " UTD-R Cell's Library," United Technologies Microelectronics Center, 1987.
24. R. Wilson, "Designers debate advantages of EDACs in small system," *Computer Design*, October 1, 1988.
25. W. Berry, "Designing Testable CMOS Gate Arrays," United Technologies Microelectronics Center, Publication 9400.
26. P. H. Bardell, W. H. Savir, "Built-in Test for VLSI/Pseudorandom Techniales," John Willey & Sons, 1987.

Vita

The author, Kuo-yeang Tsai, was born on January 10, 1960 in Taiwan. He received his Bachelor of Science degree in Electronics Engineering from National Chiao Tung University, Hsinchu, Taiwan in June, 1983.

After graduation from college, he was enlisted as a second lieutenant in the China Air Force and served as a platoon leader for about two years. Thereafter, he joined Microtek International Inc., Hsinchu, Taiwan as a hardware design engineer of gray-scaled image scanners: MSF-300G and MSF-400G.

He has been a graduate student at Virginia Polytechnic Institute and State University, Blacksburg, Virginia since September, 1987. He is currently employed by Digital Equipment Corporation, Hudson, Massachusetts, in their Semiconductor Engineering Group.

He is a member of the IEEE and the IEEE Computer Society.

Reading/writing New Chinese poetry and playing tennis are his hobbies.