# A FAULT TOLERANT BUS INTERFACE UNIT BASED ON THE NUBUS STANDARDIZED BUS ARCHITECTURE

by

Prasad Govind Paranjape

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:

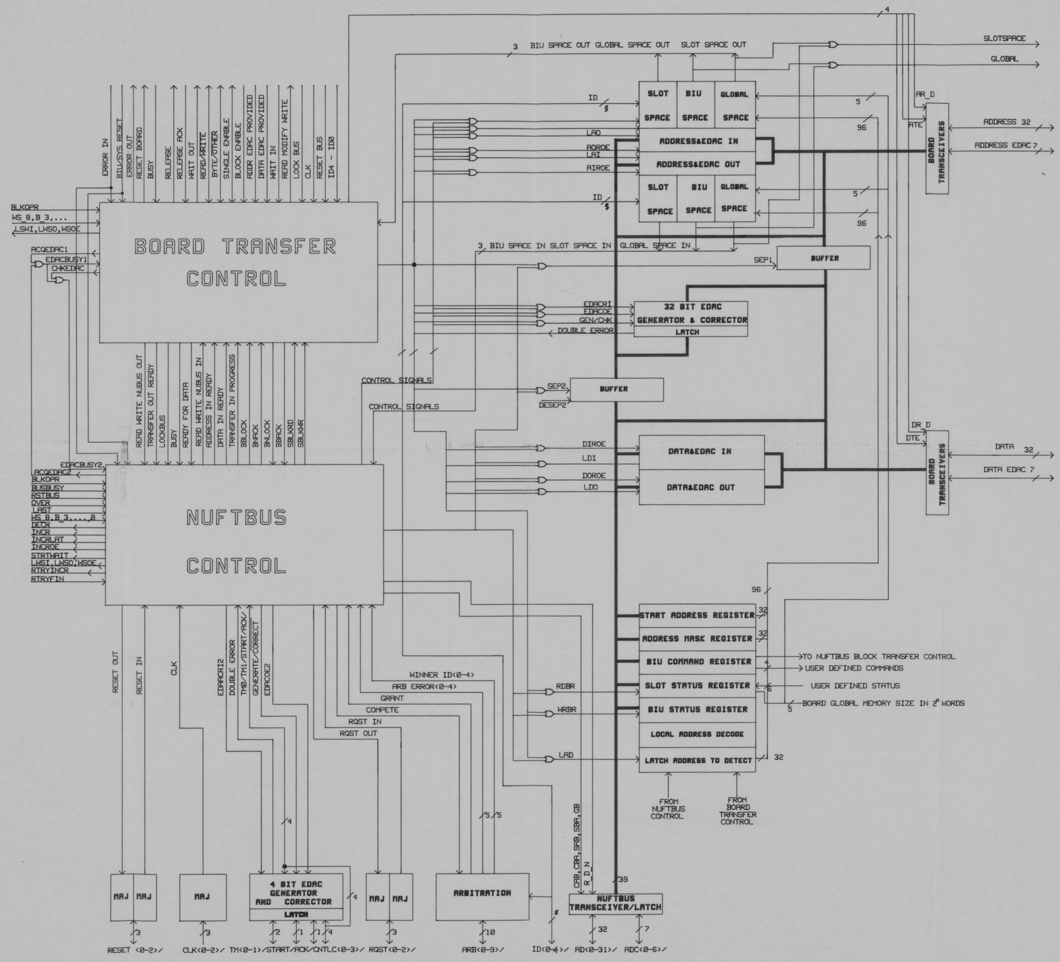
Dr. J. G. Tront, Chairman

Dr. J. R. Armstrong

Dr. S. F. Midkiff

January, 1988

Blacksburg, Virginia


BOARD TRANSFER CONTROL
NUFTBUS CONTROL
ERROR IN
BIU/SYS RESET
ERROR OUT
RESET BOARD
BUSY
RELEASE
RELEASE ACK
WAIT OUT
READ/WRITE
BYTE/OTHER
SINGLE ENABLE
BLOCK ENABLE
ADDR EDAC PROVIDED
DATA EDAC PROVIDED
WAIT IN
READ MODIFY WRITE
LOCK BUS
CLK
RESET BUS
ID4 - ID0
BLKOPR
WS_8,B_3,...
LSWI,LWSO,WSOE
ACQEDAC1
EDACBUSY1
CHKEDAC
READ WRITE NUBUS OUT
TRANSFER OUT READY
LOCKBUS
BUSY
READY FOR DATA
READ WRITE NUBUS IN
ADDRESS IN READY
DATA IN READY
TRANSFER IN PROGRESS
BBLOCK
BNACK
BNLOCK
BBACK
SBLKRD
SBLKWR
EDACBUSY2
ACQEDAC2
BLKOPR
BUSBUSY
RSTBUS
OVER
LAST
WS_8,B_3,...,8
DECR
INCR
INCRLAT
INCROE
STRTWAIT
LWSI,LWSO,WSOE
RTRYINCR
RTRYFIN
3 BIU SPACE OUT GLOBAL SPACE OUT SLOT SPACE OUT
4
SLOTSPACE
GLOBAL
ID 5
SLOT SPACE
BIU SPACE
GLOBAL SPACE
LAO
AOROE
LAI
AIROE
ADDRESS&EDAC IN
ADDRESS&EDAC OUT
5
96
AR_D
ATE
BOARD TRANSCEIVERS
ADDRESS 32
ADDRESS EDAC 7
3 BIU SPACE IN SLOT SPACE IN GLOBAL SPACE IN
SEP1
BUFFER
EDACRI
EDACOE
GEN/CHK
DOUBLE ERROR
32 BIT EDAC GENERATOR & CORRECTOR
LATCH
CONTROL SIGNALS
CONTROL SIGNALS
SEP2
DESEP2
BUFFER
DR_D
DTE
DATA 32
DATA EDAC 7
DIROE
LDI
DOROE
LDO
DATA&EDAC IN
DATA&EDAC OUT
96
START ADDRESS REGISTER
ADDRESS MASK REGISTER
BIU COMMAND REGISTER
SLOT STATUS REGISTER
BIU STATUS REGISTER
LOCAL ADDRESS DECODE
LATCH ADDRESS TO DETECT
32
TO NUFTBUS BLOCK TRANSFER CONTROL
USER DEFINED COMMANDS
USER DEFINED STATUS
BOARD GLOBAL MEMORY SIZE IN $2^N$ WORDS
RDBR
WRBR
LAD
FROM NUFTBUS CONTROL
FROM BOARD TRANSFER CONTROL
RESET OUT
RESET IN
CLK
EDACRI2
DOUBLE ERROR
TM0/TM1/START/ACK/
GENERATE/CORRECT
EDACOE2
WINNER ID<0-4>
ARB ERROR<0-4>
GRANT
COMPETE
RQST IN
RQST OUT
CAB,CBA,SAB,SBA,GB
R_D_N
39
MAJ
4 BIT EDAC GENERATOR AND CORRECTOR
LATCH
ARBITRATION
NUFTBUS TRANSCEIVER/LATCH
RESET <0-2>/
CLK<0-2>/
TM<0-1>/START/ACK/CNTLC<0-3>/
RQST<0-2>/
ARB<0-9>/
ID<0-4>/
AD<0-31>/
ADC<0-6>/

# A FAULT TOLERANT BUS INTERFACE UNIT BASED ON THE NUBUS STANDARDIZED BUS ARCHITECTURE

by

Prasad Govind Paranjape

Dr. J. G. Tront, Chairman

Electrical Engineering

(ABSTRACT)

Microprocessor based systems are used for a variety of applications, ranging from industrial control systems to spaceborne systems. The complex nature of tasks to be performed has led to division and distribution of work among different subsystems. A fast and reliable means of information and data transmission among these subsystems is provided by parallel communication busses.

Satellite-based systems are susceptible to transient faults caused by cosmic radiation or alpha particles. In order for a system to be usable in such an environment, it must be designed to be upset tolerant. Functionality of the design must to be intact in the presence of transient faults.

Several standardized bus architectures have been configured to meet a given set of performance specifications. One such bus architecture called the Nubus is used as the basis for the design and development of an upset tolerant bus architecture. The modified structure is called NuFTbus for Nu Fault Tolerant bus. Rationale for the NuFTbus specification is presented in this thesis. A design of an IC-based bus interface unit is developed. The design is specified in the VHSIC Hardware Description Language (VHDL) and VHDL tools are used to simulate the system behavior. Simulation results are presented. The VHDL circuit description is converted to a gate array layout ready for fabrication in an appropriate radiation hardened gate array technology. A description of the hardware functional testing facilities, along with a description of a set of test procedures, is given.

# Acknowledgements

I wish to express my sincere gratitude and appreciation to whose constant encouragement, support and guidance from the conception of this project till the end were tremendous. I have benefitted a great deal from working with him and I hope to get an opportunity to work with him again.

I would like to thank for his contribution during the meetings of the NRL research group, and Dr. Midkiff for his time and attention as a member of my graduate committee. I would also like to thank of for their invaluable suggestions during the development of the project.

I would like to express my gratitude to all my friends for their constant support and encouragement and to my parents, to whom I dedicate this thesis.

# Table of Contents

# List of Tables

# List of Illustrations

# 1.0 INTRODUCTION

With the progress in Very Large Scale Integrated (VLSI) technology, microprocessors are becoming more and more powerful in their computing capabilities. Microprocessors are also becoming more flexible, without any significant increase in costs. These developments have led to the use of microprocessors and microprocessor based systems in a wide variety of applications. These applications range from industrial control systems, telephone switching systems to spaceborne systems. Usually in all such systems, different tasks are performed by different subsystems, which share resources and exchange information. These systems may be governed by a single master controller or they may operate in a multimaster environment.

The ability to transfer data and control information easily and reliably among different subsystems is an important factor in the proper functioning of the system. The transfers must be fast and reliable, keeping the integrity of data intact. This requirement necessitates the use of a bus architecture to connect the subsystems [1]. Two classes of bus architecture are serial buses and parallel buses. Currently there are numerous bus standards in use in industry [2]. Many times parallel buses are preferred over serial ones because of their higher data transfer capacity.

Coincidentally with advances in the development of microprocessors from 8-bit machines to 16 and 32-bit machines, bus architectures have also progressed a long way. At present there are at least five 32-bit parallel bus standards already in use or in the process of finalizing a protocol [3]. So the choice of an appropriate bus architecture is not a simple one. It depends upon the system environment as well as on the microprocessors used. Many of the bus standards are better suited for a particular type of microprocessor family.

Whenever applications to space-borne systems are concerned, the system must not only be functional, but it must also have some degree of fault tolerance. Space-borne 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 [4,5]. Transient faults cause no permanent damage but they can change the state of a flip flop from 1 to 0 or vice versa. These are also referred to as soft errors or single event upsets (SEU). SEUs also affect combinational logic in a circuit. An SEU in combinational logic produces a change in the voltage level of a signal. This change lasts for a very short time (of the order of nanoseconds). Thus, a pulse is created on that signal line and it propagates through the combinational logic. If the outputs of the combinational logic are not sampled, or stored into a latch in that duration, then the SEU does not have any effect on the circuit. But if the occurrence of the pulse generated by an SEU coincides with the arrival of a clock edge to a flip flop, and if the duration of the pulse meets the setup and hold time requirements for the flip flop, a wrong value can get stored in the flip flop. Thus, the chance of the occurrence of an SEU is proportional to the number of flip flops present on the chip. Susceptibility of circuits to such upsets leads to the incorporation of upset detection, location and correction mechanisms in various subsystems as well as in the bus architecture. The bus architecture should have error detection and correction as well as error reporting capabilities.

Most present bus architectures provide parity bits for address/data error detection at the destination location. A single bit parity check leads to detection of single bit errors. But the provision of parity bits is not enough and features need to be added to the bus protocol to make it fault tolerant. Another feature to be considered is the number of active signal lines in a bus protocol. The larger the number of active signals on the bus, the larger the complexity of the bus interfacing hardware. More hardware increases the chance of the occurrence of upsets. Thus, a bus architecture with the least number of active signal lines, and having fault tolerance features such as single bit error detection and correction is necessary.

This master's thesis work involves the selection of a bus standard, considering criteria such as number of active signal lines and fault tolerance characteristics, designing a bus interface unit in accordance with this bus standard and working towards the implementation of this bus interface unit as a VLSI chip.

The main contributions done in this work are the following:

1. Demonstration of the concept
2. Incorporation of fault tolerant features in the modification of the parallel bus and in the design of the bus interface unit
3. Design and verification of the arbiter for the parallel bus (The design and the simulations are explained in Chapters 5 and 6, respectively).
4. Design and verification of the state machines for the control units of the bus interface unit (The design and the simulation results are explained in Chapters 4 and 6, respectively).

Different bus architectures are presented and a modified version of Nubus architecture is chosen as the basis for the bus interface unit. Features are added to the basic Nubus signals to make it more fault tolerant. Nubus is used by Texas Instruments, Inc. (where the Nubus originated) for their internal products [6]. It is also used by other computer manufacturers like Apple Computers, Inc.[7].

The proposed design of the bus interface unit differs considerably from the one implemented in Apple Computers Macintosh II. Macintosh II has divided the BIU control unit into three state machines [7]. CPU→Nubus state machine carries out all the handshake from CPU to the Nubus (essentially, when the bus interface unit is in the master mode). Nubus→CPU state machine handles the slave transactions to the CPU side and the third state machine handles the Nubus transactions in the slave mode. The proposed BIU has two state machines, one handles all the protocols to and from the CPU side of the BIU while the other control unit handles all the protocols to and from the Nubus side of the BIU. Also, block transfers are implemented in this design while Apple's design does not allow for block transfers.

## 1.1 Organization of thesis

This thesis is written to document the design and development of the fault tolerant version of the Nubus BIU. Chapter 2 describes three common 32-bit bus architectures and justifies the choice of the Nubus. Chapter 3 lists the modifications made to the Nubus standard to incorporate a degree of fault tolerance. Chapter 4 explains the operation of the BIU in detail with reference to the Nubus protocols, such as single word read, single word write, block read and block write.

Chapter 5 gives the detailed design of the BIU. Chapter 6 explains the simulation carried out on the different components of the BIU and gives the simulation results. Chapter 7 explains the initial steps in the implementation procedure necessary to fabricate the BIU as a CMOS VLSI chip. Chapter 8 gives the testability features which can be incorporated in the BIU in order that the chip can be tested after fabrication. Chapter 9 summarizes the development process and gives direction to future work in this area.

# 2.0 SELECTION OF NUBUS

## 2.1 Introduction

The most popular 32-bit buses at present are Multibus II, VMEbus and Nubus. There is no industry-wide standard for 32-bit buses so far, so each bus's proponents are making efforts to produce a de facto industry standard [8].

Multibus II is sponsored and used by Intel. VMEbus is sponsored by Motorola and supported by Signetics and Mostek. Nubus is sponsored by Texas Instruments and supported by Lisp Machines, Inc. and Apple Computers. Multibus II and Nubus are synchronous buses while VMEbus is an asynchronous bus. All three bus standards are described in the following sections. The description includes important characteristics of each bus, of sub-buses, if any, and of arbitration and interrupt mechanisms.

## 2.2 VMEbus (IEEE P1014)

VMEbus is an asynchronous parallel bus developed from Versabus, a backplane bus for initial MC68000-based systems. It provides over 4 Mbytes as primary address space and over 4 Gbytes as secondary address space. It also has 64 Kbytes of input/output space. The total number of active signals on the VMEbus is 107 [9].

### 2.2.1 VMEbus Architecture

Eight different functional modules are supported by the VMEbus standard. They are briefly described as follows [10].

- Master Module: The master controls the transaction on the DTB (Data Transfer Bus) between itself and its slave. Before starting the transaction, the master requests control of the bus through the bus requester. The master is generally a CPU but can also be a control unit that is capable of performing data transfers on the bus (e.g., a DMA controller).

- Slave Module: The slave responds to the transaction initiated by a master after decoding the address lines, the address modifier lines and the strobe signals. A slave can be a memory or an I/O device or another CPU module.

- Bus Requester: The bus requester requests control of the data bus for potential masters or for an interrupt handler. A bus requester is associated with each master or with the interrupt handler. Bus requesters are of two types, RWD (Release When Done) and ROR (Release On Request). Generally, RWD bus requesters are associated with DMA controllers while ROR are associated with masters which need the bus more often.

- Bus Arbiter: This is the central arbiter for the system. It arbitrates the bus requests and assigns control to one of the masters based on the arbitration algorithm selected. Requests can be made on four levels (irrespective of the type of arbiter). Within each level any number of sublevels can exist. The sublevels are daisy chained. A single level arbiter is associated solely with the highest priority transactions. A fixed priority arbiter arbitrates among all four levels and gives control to the highest priority requester. A round robin arbiter gives equal access to all levels.

- Interrupter: The interrupter generates an interrupt upon receiving a request from a local module. There are seven levels on which requests can be made. The levels are selectable by hardware jumpers or by software. The interrupt acknowledge signal is daisy chained from one interrupting device to the next. Upon receiving an acknowledgement, the requesting interrupter responds by providing an 8-bit vector.

- Interrupt Handler: Once the interrupt handler receives an interrupt, it arbitrates and receives control of the bus through the bus requester. Next, it sends an acknowledgement on one of the seven levels to the requesting interrupter. The acknowledgement level is selected by a 3-bit code.

- System Clock Module: This module provides a 16 MHz clock to all of the modules on the bus. Transactions on the bus are asynchronous but the modules may derive internal timing using the system clock.

- Power Module: This module provides +5, +12, -12 V to the system. It also provides Reset and Power Fail signals.

### 2.2.2 Sub-buses

The VMEbus standard defines four sub-buses. They are:

- Data Transfer Bus (DTB): Data Transfer Bus consists of the following signals:

- 32-bit address and 32-bit data path (non multiplexed)
- Address Modifier lines
- Control lines (WRITE*, IACK*, AS*, etc.)

The data path is justified. That is, partial-width data, such as bytes or half words, are moved from their original position so that they occupy the least-significant signal lines [2]. Transfers of size 8, 16 or 32-bits can be made at a time. Also the address path can be 16, 24 or 32-bits wide. Address Modifier bits ($AM_5$-$AM_0$) define 64 data transactions. Fourteen different data transactions are defined by the IEEE standard P1014, sixteen can be user defined and the rest are reserved for future use.

- Arbitration Bus: The arbitration bus consists of the following signals:

  - Bus Request (4)
  - Bus Grant In (4)
  - Bus Grant Out (4)
  - Bus Busy and Bus Clear

  The four lines used in each group (request, grant in and grant out) allow for four arbitration levels. Requests can be made on any one of the four lines. Grant In and Grant Out lines are daisy chained.

- Interrupt Bus: The signals on the interrupt bus are:

  - Interrupt Request (7)
  - Interrupt Acknowledge (IACK)
  - Interrupt Ack In (IACKIN)

- Interrupt Ack Out (IACKOUT)

Seven interrupt request lines are used to produce a seven level interrupt structure. IACK differentiates the interrupt acknowledgement cycle from the data transfer cycle. IACKIN and IACKOUT are daisy chained.

- Utility Bus: The utility bus contains all of the supporting lines such as system clock, reset, system fail, AC power fail and power and ground buses.

VMEbus requires two 96-pin connectors. The first connector supports a subset of VMEbus with a 16-bit data and a 24-bit address path (to go along with 16-bit processors like the Motorola's MC68000). The second connector which acts as an extension, makes both address and data paths 32-bits wide. (All of the control signals are on the first connector).

## 2.3 Multibus II

Multibus II is a redesign of Multibus (IEEE 796) to a incorporate 32-bit wide data path. Multibus II has five sub-buses, namely

- iPSB (parallel system bus)
- iLBX (local bus extension)
- iSSB ( serial system bus)
- iSBX (I/O expansion bus)
- Multichannel DMA I/O bus.

Functions can be carried out in parallel on all of the sub-buses of the Multibus II, provided that the system architecture is such that it takes advantage of this bus structure. The following section describes these sub-buses briefly [11].

- iLBX: iLBX can be used to expand the local memory to 64 Mbytes. It is a synchronous bus with a maximum clock frequency of 12 MHz. iLBX is processor independent and supports 8, 16 and 32 bit-processors and up to 6 boards. It has a bandwidth of 48 MHz. iLBX supports block transfers but does not have any input/output or message passing capabilities.

- iSSB: This bus is a 1-bit serial bus running at 2 MHz. It implements a message passing protocol.

- iSBX: iSBX is a carry-over from the Multibus I architecture. It allows on-board system expansion using small multimodule boards. Use of iSBX I/O expansion bus alleviates the cost of adding another full expansion board.

- Multichannel DMA I/O bus: This facility supports high speed (8 Mbytes/sec) block transfers of data between peripherals and computer boards. As the name suggests, this bus is mainly used for DMA operations. This bus is also a carry-over from Multibus I.

- iPSB: The parallel system bus, being the main system bus, is explained in detail. iPSB has a 32-bit multiplexed address/data path giving it access to a 4 Gbytes address space. It is a synchronous bus with a maximum clock frequency of 10 MHz. The bus operations are defined in terms of bus cycles. The bus cycles are:

- Arbitration cycle: In the arbitration cycle, the modules compete to get control of the bus (here, bus refers to iPSB). There are two phases in the arbitration cycle, a resolution phase and an acquisition phase. In the resolution phase, all of the modules arbitrate and the bus master is selected by a distributed arbitration logic. In the acquisition phase, the new master begins its transaction while other modules which did not win last time, start the resolution phase for the next arbitration cycle.
- Transfer cycle: The transfer cycle can also be divided into two phases, a request phase and a reply phase. In the request phase, the bus master places the address and the control information on the bus. The control information defines the type of transaction and the type of address space to be acted on. In the reply phase, a slave responds to the master and a data transfer may be carried out depending upon the type of the transaction. The end of the reply phase is indicated by an EOC signal. Once the master gets EOC, it releases the bus mastership and a new bus master takes control of the bus.
- Exception cycle: If a module detects an exception in a transfer cycle, it generates an exception indication after which all the modules begin an exception cycle. An exception cycle can terminate both arbitration and transfer cycles.

The signals on PSB are divided into 5 groups as follows:

- The address/data signal group consists of multiplexed address/data lines and four parity lines. Each parity bit generates an even parity for one byte of address/data.

- The arbitration cycle signal group consists of arbitration ID lines and a bus request line. All of the requesting modules assert the bus request line in the resolution phase of the arbitration cycle and put their ID code on the arbitration ID lines. Bus request and arbitration ID lines are wire-ORed together.

- The system control signal group provides control and handshake signals. There are 10 system control signals $SC_9$-$SC_0$. These signals define the type of transaction being performed and give other control information in the request phase of the transfer cycle while they report status information in the reply phase of the transfer cycle. Some of the functions of the SC lines are bus locking indication, EOC indication, and error indication. $SC_8$ and $SC_9$ are parity lines for $SC_7$-$SC_4$ and $SC_3$-$SC_0$, respectively.

- The exception cycle group consists of BUSERR and TIMEOUT signal lines. BUSERR indicates a parity error on the address/data lines or $SC_9$-$SC_0$ lines. The TIMEOUT line is activated by the CSM (Central Service Module) which acts as a bus monitor. The TIMEOUT line is activated when CSM determines that a slave is taking too long to respond to a handshake signal.

- The utility group (central control signal group) consists of seven signals such as clock, reset, power low.

The arbitration logic is distributed among all system modules and fully supports priority acquisition on 32 levels. Arbitration can be performed at normal priority or at high priority. The Multibus II system architecture divides the system's modules into two types. The modules which operate at higher priority make their $ARB_5$ line active (i.e., low) while the modules which are to operate at normal priority have their $ARB_5$ line equal to one. The modules drive the ARB lines with their board ID. The ARB lines are wire-ORed together. Thus, at the end of the arbitration cycle if the $ARB_5$-$ARB_0$ lines match a module's ID code, that module becomes the new master. The arbitration mechanism in the two types of modules, normal priority modules and high priority modules, is explained as follows:

- Normal priority: If no high priority module is contesting for the bus, then this scheme employs fairness in the arbitration. That is, modules are allowed to enter the contest only when no bus request cycle is ongoing. Thus, a new module cannot start to contest for the bus unless all the modules in the previous round of arbitration receive their turn as bus masters.

- High priority: If a high priority module wants to take control of the bus in the middle of an arbitration cycle, it places its ID code on the arbitration lines. When the current bus transaction is over and a new resolution phase is entered, the high priority module competes with the remaining modules. In this resolution phase, the high priority module wins the arbitration contest over any other normal priority modules. If there is another high priority module in the contest, then the contest is settled depending upon the modules' respective ID codes. The module with the lower ID code wins the arbitration contest.

Multibus II makes provision for message passing. A dedicated address space is defined by iPSB for passing messages between modules. Interrupts are supported through messages. These messages do not have a data field but consist of an 8-bit ID representing the module sourcing the interrupt and the 8-bit ID of the module that should respond to that particular interrupt.

Multibus II has in all 62 active signal lines. A 96-pin DIN connector is prescribed for signals plus power and ground.

## 2.4 Nubus

The Nubus is a synchronous system bus with multiplexed 32-bit address/data path. Though synchronous in nature, the Nubus uses simple handshake protocols which enable the transactions to vary in length (i.e., the number of clock cycles). This variation gives the adaptability of an asynchronous bus without losing the design simplicity of a synchronous bus. The basic objectives of the Nubus are [12]:

- Optimized bus for true 32-bit transfers
- System architecture independence
- Multiprocessor support
- Ease of system integration

The designers of the Nubus have tried to make the protocol as simple as possible. Instead of using multiple specialized buses, the Nubus uses a single synchronous parallel bus. The peak data transfer rate of the Nubus is 37.5 Mbytes/sec. It supports only read and write transactions (single word transfers as well as block transfers). The handshake protocol is simple and synchronized with the bus clock. The Nubus has just 51 active signal lines. It can support up to 16 modules on a backplane. Any module can arbitrate for the bus and can serve as the bus master. Module addressing is geographical depending on the slot position in which a module resides. Each slot has an ID code hardwired into it, which obviates the use of jumpers or switches on the module. The block diagram of a typical Nubus system is shown in Figure 1. The address space of the Nubus is 4 Gbyte. Arbitration is distributed like that of the Multibus II but unlike the arbitration of the Multibus II, Nubus enforces fairness in sharing of the bus bandwidth. Fair arbitration means that if a higher priority module requests the bus while an arbitration contest is going on, that module will get control of the bus only after all the modules already competing for the bus get control of the bus and finish their transactions. Thus, after requesting the bus, every module irrespective of its priority will get control of the bus within a certain time period. Arbitration takes place simultaneously with a data transaction [13].

### *2.4.1.1 Linear Address space*

The Nubus supports over 4 Gbytes of address space. The physical address ranges from 00000000H to FFFFFFFFH. All transactions are carried out using linear physical addresses. Therefore virtual to physical address translation is not required. The upper one-sixteenth of the address space (256 Mbytes), called the slot space, is shown in Figure 2. Slot space is divided into 16 slots, each of 16 Mbytes. Each address slot is mapped into 16 possible Nubus card slots (defined by the 4-bit ID code). The slot space for a module with slot ID $S_i$ , ($0000 \leq S_i \leq 1111$) is F$S_i$000000H to F$S_i$FFFFFFH. As the slotspace is determined by the slot ID of the module, the use of jumpers or configuration registers is avoided.

The rest of the address space (3840 Mbytes) is referred to as global address space. Allocation of part of the global memory to individual modules is system dependent. The allocation is done by programming a register in the slot space of each module. Interrupts

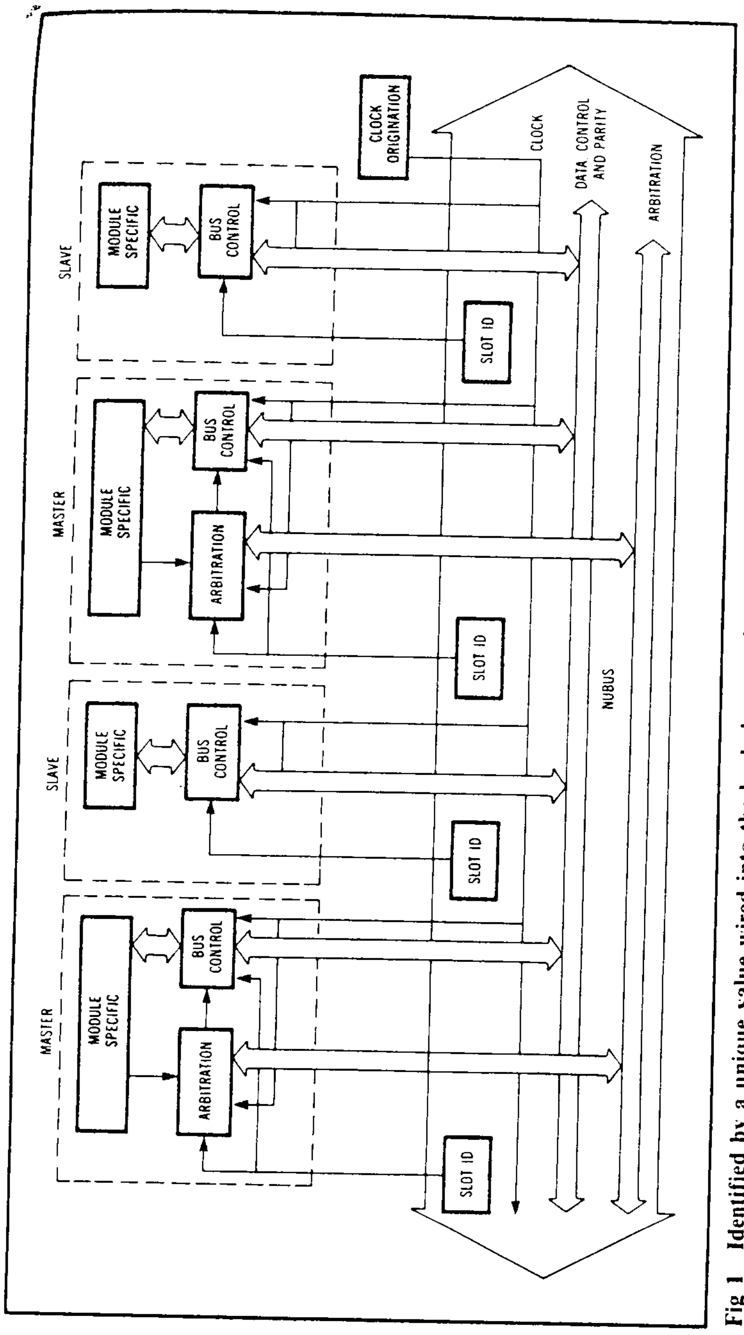


**Fig 1 Identified by a unique value wired into the backplane, each module capable of participating in the arbitration mechanism is free to serve as either bus master or slave in the NuBus. A 10-MHz system clock synchronizes this 32-bit multiplexed bus to provide up to a 37.5-Mbyte/s transfer rate in block mode.**

Figure 1. Block diagram of a Nubus system

and input/outputs are implemented by taking advantage of the linear physical address space.

Interrupts are implemented as write transactions and require no unique protocol or signals. The write transaction which is used to post interrupts is called an event transaction. A module can interrupt another module on the Nubus by doing a write operation into a particular area of the slot space which is monitored by that processor. Any address in the slot space can serve as the interrupt space. The predefined address space can either be polled by the processor or the address may be hardwired to generate an actual interrupt to the processor. In this way the interrupts can be posted to individual processor modules.

The Nubus supports unjustified data transfers. So a transaction reading/writing byte n ($0 \leq n \leq 3$) actually gets byte n from the bus data path.

Four types of bus cycles are supported by the Nubus. They are start cycle, data cycle, ack cycle and attention cycle. When a master is ready to start its transaction, it sends out the address and control information on the bus in a start cycle. The start cycle is followed by single/multiple data cycles and an ack cycle. The slave responding to the transaction initiates the ack cycle and passes status information back to the master.

The attention cycle is a single bus cycle in which both the start and ack lines are asserted by the master. $TM_1$ and $TM_0$ indicate the type of attention cycle. The attention cycle can be used for reinitializing the arbitration process or for resource locking. The other two possible cycle types are reserved for future use.

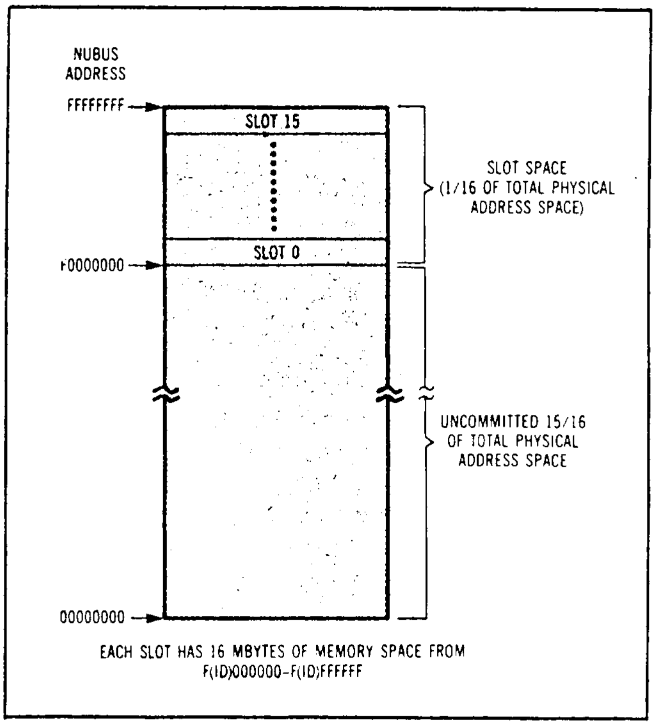

NUBUS ADDRESS
FFFFFFFF
SLOT 15
SLOT 0
F0000000
00000000
SLOT SPACE (1/16 OF TOTAL PHYSICAL ADDRESS SPACE)
UNCOMMITTED 15/16 OF TOTAL PHYSICAL ADDRESS SPACE
EACH SLOT HAS 16 MBYTES OF MEMORY SPACE FROM F(ID)000000-F(ID)FFFFFF

Figure 2. Allocation of memory of the Nubus system

The Nubus signals are divided into four groups. The groups are:

1. Utility signals: Utility signals consist of RESET, CLK, PFW (power fail warning), card slot ID (these are not bussed but are binary coded at each slot to specify the module position on the backplane), and NMRQ (non-master request is an asynchronous line asserted by the boards which are not capable of becoming masters, to indicate their need to communicate with the master).

2. Bus data transaction signals: Bus data transaction signals are tristatable. They include Address/Data lines, $AD_{31}$-$AD_0$ (32-bit multiplexed), four control lines $TM_1$, $TM_0$, START and ACK, system parity SP and parity valid SPV. In the start cycle, $TM_1$ and $TM_0$ along with $AD_1$ and $AD_0$ indicate the type of transaction (single word read/write, block read/write or half word read/write), while in the ack cycle, $TM_1$ and $TM_0$ give the status information of the transaction which was just completed. The status can be: transfer complete, error, timeout or try again later.

3. Arbitration System Signals: Arbitration system signals are open collector lines and are used by the distributed arbitration logic to determine the next master of the bus. These signals include Bus Request (RQST) line and arbitration signals ($ARB_3$-$ARB_0$). The arbitration is strictly fair. All of the modules which need access to the bus, check the RQST line. If RQST is deasserted, it is asserted by the requesting modules. The requesting modules place their respective ID codes on the ARB lines and start contesting for the bus. As the arbitration is local, the module with the highest ID code gets a local GRANT signal indicating that it is the new bus master. The RQST line can also be used for bus locking.

4. Power lines: Four power supply voltage levels are defined by the Nubus: +5V, -5.2V, +12V, and -12V.

Nubus uses a 10 MHz clock with a duty cycle of 25%. The rising edge of the clock is the driving edge, and the falling edge is the sampling edge (see Figure 3). The 75 ns between the driving and the sampling edges allow for propagation delays and setup timing, while the 25 ns between the sampling and the driving edges avoid clock skew problems [14].

## 2.5 Comparison of the buses

This section compares the three bus architectures discussed earlier and justifies the choice of the Nubus over the other buses.

1. System architecture independence

   Both the VMEbus and the Multibus use specialized multiple buses for DMA transfers, exception handling and input/output operations. These buses tend to create their own system architecture philosophy. Even though the Multibus and the VMEbus are said to be processor independent, the bus is conceived for, and optimized around only one particular family of processors. The processor dependence is also caused by specific bus signals, their timings, justification of data paths and nonuniform address spaces.

   On the other hand the Nubus, with only one parallel system bus, simplifies the bus to board handshake and thus is truly system architecture independent.

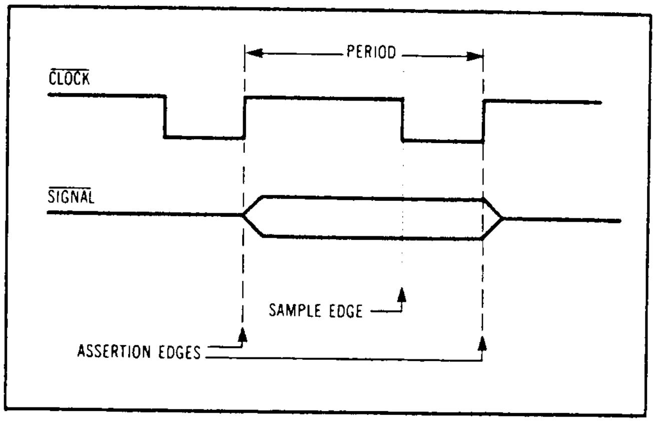


Figure 3. Driving and sampling edges on the Nubus

2. One linear address space

As explained in the previous section, the Nubus supports a single linear address space with fixed slot spaces linked to each module and a shared global memory. The advantage of this scheme is that the system programmer can always access a physical address within memory, can make input/output references, or can initiate interrupt operations. Thus, one address space serves all the above mentioned purposes. This simplifies the programming work considerably. Also the ability to post interrupts to individual processors facilitates a multiprocessing environment.

3. Number of active signals

The Nubus protocol is simple and requires the smallest number of active signal lines (only 51) as compared to the other buses. The simplified bus structure leads to less hardware and reduced complexity of design and effectively reduces possible fault insertion areas.

4. Synchronous bus vs. asynchronous bus

There is no indisputable resolution to the argument as to whether to choose a synchronous or an asynchronous bus. Some of the benefits of a synchronous bus are [6,15]:

- Higher noise immunity: Synchronous buses are said to have better noise immunity, as noise can be a problem only when it occurs in a very small time frame. That is, noise can only affect the system when signals are sampled/latched by the clock pulse.

- Simplicity: Synchronous buses require fewer handshake signals (the VMEbus uses 107 active signals while the Nubus uses only 51 active signals), so it is easier to design around a synchronous bus structure. Though there is a theoretical upper bound on the bandwidth of a synchronous bus (dependent on the system clock), a clock of 10 MHz which is commonly used for synchronous buses is high enough to support at least another generation of IC technology [6].

These points justify the selection of a synchronous bus.

5. Fair arbitration

The Nubus enforces fairness in the arbitration mechanism which is very important in the case of multiprocessing systems. Every requesting module is guaranteed to obtain bus mastership within a certain maximum time.

These factors support the choice of the Nubus as the 32-bit parallel system bus over the Multibus II and the VMEbus for this application.

The next chapter explains modification of the Nubus necessary to incorporate a high degree of fault tolerance features.

# 3.0 NUFTBUS : MODIFICATION OF NUBUS

## 3.1 Introduction

This chapter deals with the necessity of incorporating fault tolerance features into the system bus and with the modifications made to the Nubus specification to make it fault tolerant. A measure of the degree of fault tolerance achieved by these modifications is given at the end of the chapter.

## 3.2 Necessity of a fault tolerant bus

For all computing systems, reliability and incorporation of fault tolerance features are important for safety as well as for the proper functioning of the system. These tasks become more important when a computer system is to be stationed in a satellite or other spacecraft. These systems must be able to operate correctly, even when unattended for long periods of time.

The parallel system bus is an important part of any computer system. The main function of such a bus is to exchange commands, messages or data between different processors in a multiprocessing environment. Thus, it is necessary that the transfers 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 fault 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 next section lists the modifications/additions to the Nubus specification to make it fault tolerant. The modified Nubus is referred to as the NuFTbus (Nu Fault Tolerant bus) [16].

## 3.3 Modifications to the Nubus

### 3.3.1 ID lines

The Nubus system architecture supports 16 modules on the bus. The modules are given unique 4-bit ID codes ranging from 0000 to 1111. The ID code is used in identifying the address range assigned to each module and is also used in the arbitration process. If a stuck-at fault occurs on one of the ID lines, the ID code for that module gets changed. For example, if a module with ID code 1101 has a stuck-at 0 fault on line 2, the ID code changes to 1001. 1001 is also a valid ID code. If another module with the same ID code of 1001 does not exist in that particular system, this bit error will not cause any problems in the arbitration process (this module will simply shift its priority for obtaining the bus). However, this fault will affect the module's address decode logic. On the other hand, if a module with the ID code of 1001 does exist in the system, then both the modules will win the arbitration contest at the same time and will try to start a transaction simultaneously. This leads to a bus error and would also lead to an address space overlap.

To avoid the occurrence of this type of situation, a fifth bit (parity bit), is added to the ID code. The addition of the parity bit to each ID code 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. Only if two bits get changed will the new ID code match some other valid ID code. For the same example considered before, with the addition of a parity bit, the two ID codes will now become 11011 and 10010. So even if the second digit of the first ID code gets complemented to 0, the new ID code is 10011, which is still distinct from the other ID code. These distinct ID codes prevent two or more modules from becoming the bus masters at the same time.

## 3.4 Arbitration logic

Arbitration is a critical point as far as multiprocessing systems are concerned. Presence of a stuck-at fault on the arbitration lines ($ARB_4$-$ARB_0$) might lead to a system without a master (none of the modules wins the arbitration) or a system with more than one master. Both situations must be avoided.

The Nubus uses a distributed arbitration scheme. Distributed arbitration reduces the probability of failure as compared to a central arbiter scheme, because a distributed arbitration scheme has no single point of failure which would affect the whole system. However, stuck-at faults can affect distributed arbitration. If an ARB line is stuck-at 1, only eight modules out of sixteen can win the arbitration at any given time. This is also true for a stuck-at 0 ARB line. Another type of error that can occur is the internal bit error. A module with an internal bit error can get a false local grant signal, while actually some other module has won the arbitration. This would again lead to a bus contention problem.

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 at the same time.

Secondly, the arbitration is carried out on two distinct sets of arbitration lines, $ARB_9$ -$ARB_7$-$ARB_5$-$ARB_3$-$ARB_1$ and $ARB_8$ -$ARB_6$-$ARB_4$-$ARB_2$-$ARB_0$. The arbiter in each module places the complement of its ID code bit at two places on the arbitration lines (bit 0 will go on $ARB_0$ and $ARB_1$ , bit 1 will go on $ARB_2$ and $ARB_3$, bit 2 will go on $ARB_4$ and $ARB_5$, bit 3 will go on $ARB_6$ and $ARB_7$ and bit 4 will go on $ARB_8$ and $ARB_9$). Redundant arbitration takes place at each bit position.

The arbiter is explained in detail in Chapter 5, herewith is a brief description. 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 (this would be due to another module placing zeros on those two 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 process is continued. So even if the module has a single bit error, it can arbitrate. It may just shift its position in the fair arbitration scheme.

## 3.5 Clock/reset lines

The Nubus needs a clock signal to synchronize transactions on the bus. To ensure the proper functioning of the clock, two methods are considered. Either the clock can be made redundant or a voting scheme can be employed (Triple Modular Redundancy). Unfortunately, it is difficult to decide when to switch to a single redundant clock.

Therefore, a voting scheme is implemented in this design. There are three clock lines on the bus. Three clock lines are generated from the same oscillator and are voted in each BIU. A redundant oscillator may be provided. A similar voting scheme is implemented for the reset line.

## 3.6 Error detection and correction coding

Along with stuck-at faults, another type of fault which is important, especially in satellite systems or spacecrafts, is a transient fault. The transient fault can change the state of a memory cell from 0 to 1 or vice versa. These faults are referred to as single event upsets [5]. To detect single event upsets, error detection and correction coding using modified Hamming code is carried out on the address/data and control lines of the NuFTbus. Check bits are added to these lines. Single bit error correction and two bit error detection requires the minimum distance between the codes to be 4. So a modified Hamming code is used [17]. The number of check bits necessary depends on the number of data bits and the error correction capability required. For a 32-bit address/data path, seven check bits are added [18]. Three check bits are added to the four 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. When starting a transaction, the BIU on the master side generates the check bits on the control and address lines. A similar operation can be carried out on the data, if it is a write operation. The BIU on the slave side verifies the check bits for the validity of the data. The BIU corrects single bit errors and gives an error indication for two bit errors. Once the transaction on the slave module is over, the slave BIU generates check bits for status lines and for the data word (if it is a read operation). The master BIU checks the validity of the status lines and the data word and takes any necessary action in response to an error indication.

Following is a list of the NuFTbus signals alongwith a brief description of each.

| | |
|---|---|
| **RESET < 2:0 >** | initializes all the modules to power-up state, the three lines are voted in the BIU |
| **CLK < 2:0 >** | 10 MHz clock, the three lines are voted in the BIU |
| **PFW** | power fail warning signal |
| **ID < 4:0 >** | ID lines, are not bussed but binary encoded at each slot location |
| **NMRQ** | non-master request, for modules which are not capable of becoming masters, to indicate need for service |
| **AD < 31:0 >** | multiplexed address/data lines |
| **AD < 38:32 >** | error detection and correction coding lines on AD < 31:0 > |
| **TM < 1:0 >** | control and status lines |
| **START** | indicates the starting of a transaction and initiates an arbitration contest |
| **ACK** | indicates the ending of a transaction |
| **CNTL < 3:0 >** | error detection and correction coding lines on control signals |
| **SP** | system parity on address/data lines |
| **SPV** | system parity valid, if asserted, indicates that a parity bit has been generated |
| **RQST** | when asserted, indicates that the module would contest in the next arbitration contest |
| **ARB < 9:0 >** | wire-ORed arbitration lines to determine the next bus master |

The NuFTbus has 73 signal lines excluding the power and ground.

## 3.7 Reasons for not using a redundant bus

A different approach to make the bus fault tolerant is to provide a redundant bus (instead of providing redundancy on particular signals). In such an environment, all of the modules switch to operate with the redundant bus when required. This method has two drawbacks:

- It doubles the number of wires on the bus. (In case of the Nubus this would add 50 to 60 lines). This complicates the system and adds extra hardware. Whenever computer systems for satellites/spacecrafts are concerned, a major point to be considered is to reduce component and silicon area as much as possible. (Increased area leads to more weight and more power requirements).
- As in the case of the clocking scheme, it is always difficult to decide when to switch to the redundant bus and to synchronize all the modules to the redundant bus.

Thus adding a full redundant bus is avoided.

## 3.8 Figure of merit of fault tolerance

This section gives a measure of the effectiveness of the modifications carried out on the arbitration scheme of the Nubus.

Due to the addition of a parity bit to the ID code and the use of two sets of arbitration lines, two modules will never win the arbitration contest at the same time. Also the bus will never be without a master (a module, which may not be the right one according to the priority, will always win the arbitration). This is elaborated in the chapter on simulation results, which deals with simulations on the arbitration logic.

In this chapter the problem is approached in a more general way. At any given time i modules, ($1 \leq i \leq 16$), can arbitrate. These i modules can be chosen from 16 possible modules. Therefore, combinatorially

$$A = \sum_{i=1}^{16} C[16,i] = 65535.$$

where A gives all possible combinations of modules taking part in an arbitration contest.
(C[16,i]: Combinations of 16 things taken i at a time).

For stuck-at fault analysis, each of the ten ARB lines is stuck-at 1 and then stuck-at 0 one by one. This leads to

$$20 \times \sum_{i=1}^{16} C[16,i] = 1310680$$

different cases.

To consider all these cases exhaustively is not possible. At a first view it may seem that the results of the arbitration will depend on the number of modules taking part in the arbitration process at a particular time. (different modules place different ID codes on the ARB lines). But this is not the case. Irrespectively of the number of modules, each module will either put a 0 or a 1 on the ARB lines. Decisions are made at every bit position of the ARB lines. A winner module is decided at a bit location where only one module places a 0 on the ARB line, and the rest of the modules (those still contesting) place a 1. This can happen at bits $id_4$ to $id_1$. (This will happen at $id_0$, which is the parity bit, only in case of a single bit error).

So, to generalize the result, all the different cases with n modules ($1 \leq n \leq 16$) can be folded down to all the cases when any two modules, i and j compete. ($1 \leq i \leq 16$, $1 \leq j \leq 16$ and $i \neq j$). i and j are varied to consider all the possible combinations.

The results are as follows: for a module $S_i$, X is the number of times it was supposed to win the arbitration. Y is the number of times it actually won the arbitration. So X - Y times some module $S_j$ won the arbitration, when $S_j$ was not supposed to win. In the standard Nubus system, for these X - Y cases there would have been no bus master or multiple bus masters. The Table 1 on page 34 shows that 59 times out of 600, (for 10% of the cases), there would not have been a master in the normal arbitration scheme. In all these cases, some other module won the arbitration in the new scheme and the bus operations continued. The detailed output showing the competing ID codes and winner modules for each case is presented as a report by the author in reference [19].

The next two chapters explain the operation and design of the bus interface unit for the NuFTbus.

**Table 1. Comparison of arbitration result with/without single bit error**

|  | winner (fault free cases) | winner (even with fault) | could not win |
|---|---|---|---|
| MODULE ID | X | Y | X-Y |
| 0000 | 75 | 75 | 0 |
| 0001 | 70 | 64 | 6 |
| 0010 | 65 | 58 | 7 |
| 0011 | 60 | 55 | 5 |
| 0100 | 55 | 48 | 7 |
| 0101 | 50 | 44 | 6 |
| 0110 | 45 | 39 | 6 |
| 0111 | 40 | 32 | 8 |
| 1000 | 35 | 32 | 3 |
| 1001 | 30 | 28 | 2 |
| 1010 | 25 | 23 | 2 |
| 1011 | 20 | 16 | 4 |
| 1100 | 15 | 15 | 0 |
| 1101 | 10 | 8 | 2 |
| 1110 | 5 | 4 | 1 |
| 1111 | 0 | 0 | 0 |

# 4.0 OPERATION OF THE BIU

## 4.1 Introduction

This chapter explains the overall operation of the bus interface unit (BIU), major functional blocks of the BIU and the processor-BIU (or board-BIU) interface. The specific functional blocks of the BIU are explained later in the chapter.

## 4.2 Overview of the BIU operation

Every module in the Nubus system is connected to the Nubus through a bus interface unit. The Nubus system has one module designated as the test/boot master (STBM) module. On power up, the STBM resets all the BIUs and loads initial control information into the command register of all the BIUs. (Once the bootup is over, if a module has a processor on board that processor can load control information in the command register of its BIU). This information may include start address and size of the global address space located on that module, the number of retries for a failed operation and other control information. Once initialized, the BIU is ready for operation.

Some of the salient features of the BIU are:

- Processor independence

- Bus transactions are transparent to the processor
- Programmable number of retries for a transaction
- Support of single as well as block transfers
- Presence of error detection and correction coding

The block diagram of the BIU is shown in Figure 4. The basic operation of the BIU is as follows:

Whenever the processor in a module issues an address on its local bus, the BIU traps 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. If the address is within the memory space implemented on the module, the BIU decodes the control signals and determines the type of operation to be carried out. The operation can be 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 Nubus 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. The processor does not know whether the destination/source was in the on-board memory space or in the global memory space outside the module. The processor does not have to generate extra control signals or do any address mapping to start a Nubus operation. Thus the processor treats on-board and off-board operations the same, which is possible due to the linear address space supported by the Nubus architecture, as explained in Chapter 2. This is how the operation of the BIU is viewed from the master side of the bus. Now consider the operation of the BIU from the slave side.

When the BIU sees an address on the Nubus, it latches the address and decodes it. If the address is that of a BIU register or is within the memory space implemented in the module, the BIU then completes that transaction as a slave and sends back an acknowledgement to the master BIU. In the slave mode, the BIU requests the local processor that it release the local bus and that it complete the transaction without involving the on-board processor.

## 4.3 Processor independence

The BIU is designed so that it can interface with any type of microprocessor with a minimum amount of interfacing hardware. (In the case of the bus interface unit for the VMEbus, interfacing with the Motorola MC68000 processor would be very simple but interfacing with the Intel 8086 processor will need more hardware. Similarly, using the bus interface unit for the Multibus II, interfacing with the Intel 8086 will need lesser hardware than interfacing with the Motorola MC68000). On the contrary, interfacing any standard microprocessor to the NuFTbus system would require similar interfacing hardware. So the NuFTbus BIU is considered to be processor independent. The BIU expects certain standard control signals from the processor. The signals are Read/Write*, byte/other*, single enable and block enable. These signals are usually generated by all the standard microprocessors. If not, they can be generated by a minimum amount of logic on the module. Figure 5shows the hardware needed for generating the control signals to interface with the Intel 8085 microprocessor.

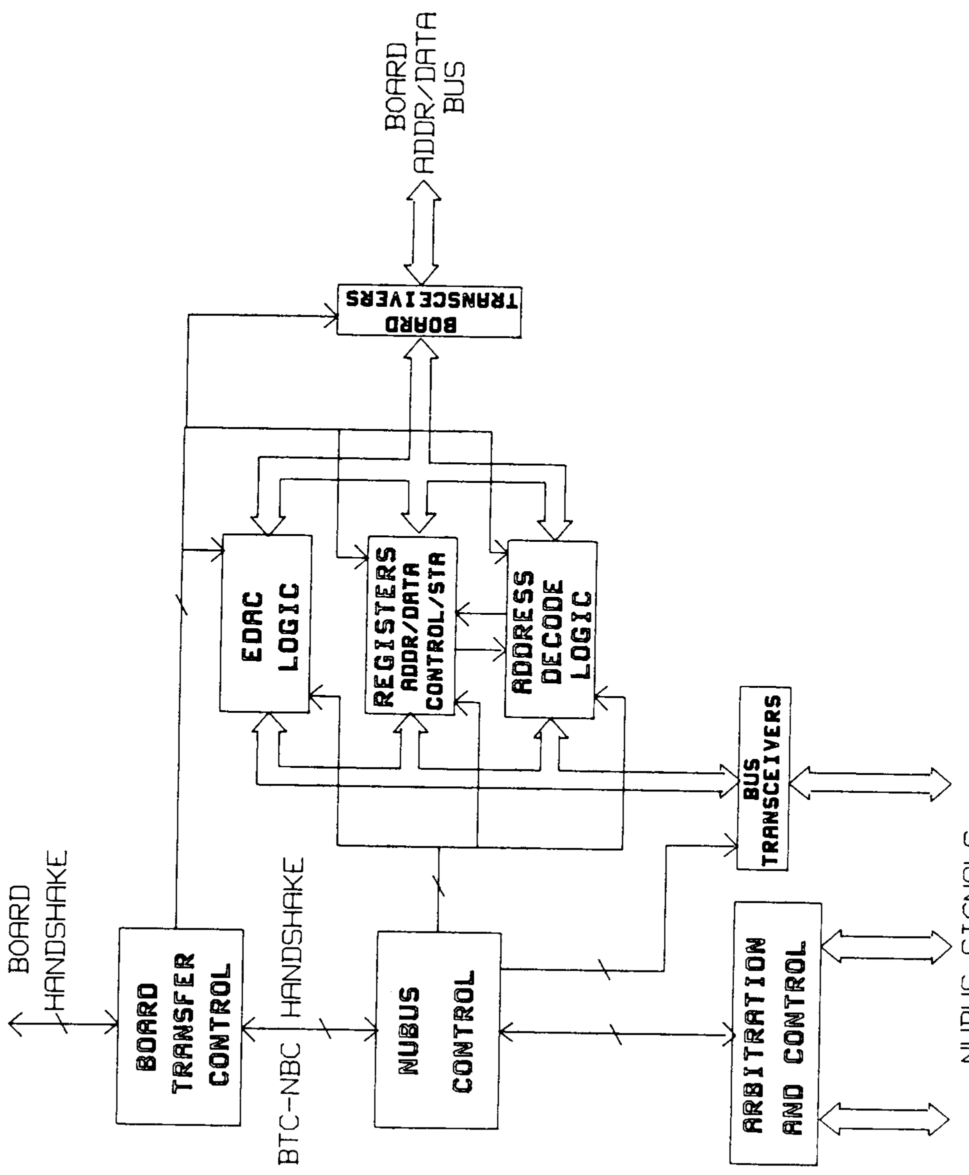


Figure 4. Block diagram of BIU

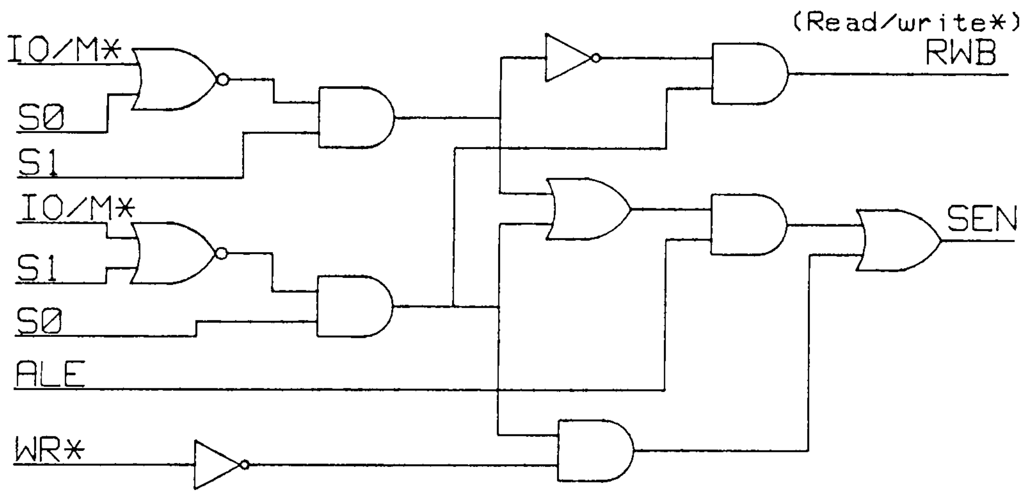


Figure 5. Generation of interface signals to BIU for an 8085 based system

## 4.4 Functional blocks of the BIU

The major functional blocks of the BIU are:

- Board Transfer Controller
- NuFTbus Controller
- Error Detection and Correction unit
- Address decode unit
- Address, Data, Control and Status Registers
- Arbitration logic
- Board and Bus Transceivers

Board Transfer Controller (BTC) is a finite state machine which performs the handshake between the processor (or the control logic on-board, if it is a memory board) and the BIU. NuFTbus Controller (NBC) is also a finite state machine which controls the arbitration logic and performs the interface operation with the NuFTbus.

There are two address decode units, one for the addresses received from the module side and the other for the addresses received from the NuFTbus side. Two address decode units are necessary, since the NBC and the BTC function as independently as possible to allow for overlap between master and slave side operations. This is explained in detail later in the chapter. 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 global memory on some other board.

There are two error detection and correction 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 of the NuFT-bus. The EDAC unit for the address/data lines is shared by BTC and NBC.

The BIU contains the following registers:

- Data In Register (DIREG): to store the data coming from the NuFTbus.
- Data Out Register (DOREG): to store the data going to the NuFTbus.
- Address In Register (AIREG): to store the address coming from the NuFTbus.
- Address Out Register (AOREG): to store the address going to the NuFTbus.
- BIU Command Register
- BIU Status Register
- Board Status Register
- Reserved register (for future use)

The BIU assumes that the local processor bus is demultiplexed. There are separate address and data transceivers in the board interface. A 40-bit bus transceiver/latch interfaces with the multiplexed address/data lines on the NuFTbus.

## 4.5 Board-BIU interface

Apart from the data bus, the board-BIU interface consists of the following control signals:

1. READ/WRITE BOARD OUT (RWBO): RWBO is an input to the BIU. It indicates a read (when high) or a write (when low) operation. (RWBO corresponds to $TM_1$ signal on the NuFTbus).

2. READ/WRITE BOARD IN (RWBI): RWBI is an output from the BIU. It is used for reading from or writing to the on-board memory.

3. BYTE/OTHER (B/O*): BYTE/OTHER is an input to the BIU. It indicates whether the read/write operation is on a byte, half word or word of data or if it is a block operation. (A word is 4-byte long according to the Nubus specifications). BYTE/OTHER corresponds to $TM_0$ on the NuFTbus. This signal might not be generated as an output pin by all the microprocessors, but it can be generated using other control signals from the processor. (e.g., it can be generated by gating signals $S_0$-$S_6$, of the Intel 8086).

4. SINGLE ENABLE OUT (SENO): SENO is an input to the BIU. A negative going edge on SENO indicates the start of an operation. The BIU latches address, data and control information with this edge. After completing the operation, SENO goes high afterwhich the BIU goes to its initial state. SENO is also used in the block transfer, as explained later.

5. SINGLE ENABLE IN (SENI): SENI is an output from the BIU. SENI functions in the same way as SENO but it is driven by the BIU instead of by the processor. SENI may be used by the on-board memory decode logic in a read/write operation.

6. BLOCK ENABLE (BLKEN): BLKEN is a BIU input signal. A negative going edge on BLKEN along with a negative going edge on SENO indicates the start of a block

operation. When the intermediate word in a block is transferred, BLKEN goes high. Once the next address (and data, for the block write operation) is ready on the local processor bus, BLKEN again goes low to start the next word transfer in the block. SENO is low during the whole block transfer operation. After the last word of the block has been transferred, both SENO and BLKEN go high simultaneously to indicate the end of the block operation.

7. ADDRESS EDAC PROVIDED (AEPR): AEPR is an input signal. A high on AEPR indicates that the on-board memory has implemented the error detection and correction 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. The BIU uses AEPR to switch between the generate and the check modes of the EDAC unit.

8. DATA EDAC PROVIDED (DEPR): DEPR is an input signal. It is associated with the data word and functions in the same way as AEPR.

9. WAIT IN (WAITI): WAITI is an input. 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 go high for the duration of the operation. The BIU puts the address and data on the local bus. Then it waits for two clock cycles (50 ns) and afterwards checks the WAITI input. If WAITI is low, the slave BIU proceeds but if WAITI is high, it holds the address and data on the local bus until WAITI goes low. Thus, if the memory is fast, it need not do anything to the WAITI line. Also the BIU need not see a pulse (a high to low and then a low to high transition) on the WAITI line. This saves having to handshake with the fast memories and still allow for slower memories to be used.

10. WAIT OUT (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 while the NuFTbus transaction is going on. When the BIU receives an acknowledgement from a slave BIU, it deasserts WAITO. This indicates to the processor that the data transfer was successfully completed. In the case of a read operation, the master processor can latch the incoming data with WAITO going low.

11. READ MODIFY WRITE (RMW): RMW is an output signal. When the BIU does a byte or a half word write operation, first it reads the contents of the address where the byte or half word is to be written. Then the BIU modifies the particular byte or half word, generates check bits on the modified word and writes the whole word back. RMW is held high during this operation.

12. RELEASE (REL): REL 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 line.

13. RELEASE ACKNOWLEDGEMENT (RELACK): RELACK is an input. A high on RELACK indicates that the BIU can take control of the local bus.

14. BUSY: BUSY is an input. Busy going high indicates to the BIU that the local bus can not be released for the use of the BIU. The BIU will generate a 'Try again later' acknowledgement to the master.

15. LOCK BUS: 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 LOCK BUS signal high. With the LOCK BUS input high, the BIU does not deassert

the RQST line of the NuFTbus after the first transaction, but retains control of the bus until the LOCK BUS input is high. This signal is used when the processor does 'Test and Set' type operation on certain memory locations.

**Note:** There is a limit on the number of transactions that can be carried out in the bus lock mode. This number is programmable and is set by STBM (system test/boot master). If a processor tries to lock the bus to a number greater than the maximum allowable number, then another other BIU that is waiting for the bus may generate an attention cycle to restart bus arbitration.

16. RESET BOARD: Reset board is an active high output. It is generated by the STBM to reset a particular module.

17. RESET BUS: Reset bus is an active high input. It is generated by a processor to reset the NuFTbus system.

18. BIU/SYSTEM RESET: It is an active high input. It is generated by the processor to reset its BIU or the NuFTbus system.

19. ERROR OUT: ERROR OUT is an output. It indicates that an error occurred during a NuFTbus transaction. The error may be a double bit error identified by the EDAC unit, or 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.

20. ERROR IN: Error in is an input. It indicates a double error in the word received by the processor from the NuFTbus.

21. CLK: CLK is a clock input from the module used for the internal operations of the BIU. The clock speed is 40 MHz.

## 4.6 Read/Write Operations on the NuFTbus

This section describes single word read, single word write, block read and block write operations. Whenever a processor in a module initiates a transaction on the NuFTbus, that module is called the master module and its BIU is called the master BIU. Similarly, the module and the BIU which respond to that transaction are called the slave module and the slave BIU, respectively.

### 4.6.1 Single word read (master side)

The following is the sequence of steps that takes place in the master BIU when a single word read is performed.

1. With a negative going edge on SENO, the BIU checks the RWBO input. A high on RWBO indicates that a single read operation is to be performed.
2. The BIU latches the address and control information into appropriate registers.
3. The BIU decodes the address and determines that it should start a NuFTbus transaction for fetching data from the slave.
4. The BIU generates a high on the WAITO output for the processor.
5. The BIU generates/checks the check bits for the address word depending upon the AEPR input.

6. The BIU arbitrates for control of the NuFTbus. Once it obtains control of the NuFTbus it asserts the START signal on the NuFTbus, outputs the control information on the control lines, and the address on the address/data bus.
7. The BIU deasserts START and the address on the next bus clock and waits for an acknowledgement from the slave.
8. When the ACK input goes low, the BIU latches the status information and the data word. Using the EDAC it determines the validity of the status and data word.
9. The BIU places data on the local processor bus and deasserts the WAITO output.
10. With SENO going high the BIU returns to its initial state.

### 4.6.2 Single word read (slave side)

The following is the sequence of steps that takes place in the slave BIU when a single word read is performed.

1. When the START signal goes low on the NuFTbus, the potential slave BIU latches the address and control information and passes it through the EDAC to check its validity.
2. The slave BIU decodes the address and determines that the address is in its address space.
3. The slave BIU requests a release from the local processor. With release acknowledge, it places the received address on the local bus and generates the necessary control signals.
4. The slave BIU sets SENI low.
5. The slave BIU waits for two clock cycles and then checks the status of the WAITI input.

6. If WAITI is high, the slave BIU waits till WAITI goes low.
7. Then the slave BIU sets SENI low and latches the data input from the local bus into the Data Out Register.
8. The slave BIU checks the validity of the data word using the EDAC unit.
9. The slave BIU generates necessary status signals, asserts ACK and places the data on the NuFTbus.
10. On the next bus clock, the slave BIU deasserts ACK and returns to its initial state.

### 4.6.3 Single word write (master side)

The following is the sequence of steps that takes place in the master BIU when a single word write is performed.

1. With a negative going edge on SENO, the BIU checks the RWBO input. A low on RWBO indicates that a single write operation is to be performed.
2. The BIU latches the address, data and control information into appropriate registers.
3. The BIU decodes the address and determines that it should start a NuFTbus transaction for writing data to the slave.
4. The BIU generates a high on the WAITO output.
5. The BIU generates/checks the check bits for the address and data words depending upon the AEPR and DEPR inputs, respectively.
6. The BIU arbitrates for control of the NuFTbus. Once it obtains control of the NuFTbus, it asserts the START signal on the NuFTbus, outputs the control information on the control lines, and the address on address/data bus.

7. The BIU deasserts START and the address at the rising edge of the next bus clock and places the data on address/data bus, and waits for an acknowledgement from the slave.
8. When the ACK input goes low, the BIU removes the data word, tristates the bus and latches the status information. Using the EDAC it determines the validity of the status.
9. The BIU deasserts the WAITO output to the processor.
10. With SENO going high, the BIU returns to its initial state.

### 4.6.4 Single word write (slave side)

The following is the sequence of steps that takes place in the slave BIU when a single word write is performed.

1. When the START signal goes low on the NuFTbus, the potential slave BIU latches the address and control information and passes it through the EDAC to check its validity.
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.
3. The slave BIU requests a release from the local processor. With release acknowledge, it places the received address and data on the local bus and generates the necessary control signals.
4. The slave BIU sets SENI low.
5. The slave BIU waits for two clock cycles and then checks the status of the WAITI input.

6. If WAITI is high, the slave BIU waits till WAITI goes low.
7. The slave BIU then sets SENI high.
8. The slave BIU generates necessary status signals and asserts ACK.
9. On the next bus clock the Slave BIU deasserts ACK and goes back to its initial state.

### 4.6.5 Block read (master side)

The following is the sequence of steps that takes place in the master BIU when a block read is performed.

1. A negative going edge on SENO as well as on BLKEN, and a high on RWBO determines that a block read operation is to be performed.
2. The BIU latches the address and control information into appropriate registers.
3. The BIU decodes the address and determines that it should start a NuFTbus transaction for fetching data.
4. The BIU outputs a high on WAITO.
5. The BIU generates/checks the check bits for the address word depending upon the AEPR input.
6. The BIU arbitrates for control of the NuFTbus. Once it obtains control of the NuFTbus, it asserts the START signal on the NuFTbus, outputs other control information on the control lines, and the target address on address/data bus.
7. The BIU deasserts START and the address on the next bus clock and waits for an intermediate acknowledgement.
8. With input $TM_0$ going low (as intermediate acknowledgement), the BIU latches the status information and the data word. Using the EDAC it determines the validity of the status and data word.

9. The BIU places the data on the local processor bus and deasserts the WAITO output.
10. 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 address, goes to step 4, and continues with the next word data transfer.
11. Both SENO and BLKEN going high indicate the end of block transfer and the BIU goes back to its initial state.

### 4.6.6 Block read (slave side)

The following is the sequence of steps that takes place in the slave BIU when a block read is performed.

1. When the START signal goes low on the NuFTbus, the potential slave BIU latches the address and control information and passes it through the EDAC 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 BIU requests release of the local processor bus. Upon receipt of the release acknowledge signal, it places the address on the local bus and generates the necessary control signals.
4. The slave BIU sets SENI low.
5. The slave BIU waits for two clock cycles and then checks the status of the WAITI input.
6. If WAITI is high, the slave BIU waits till WAITI goes low.

7. The slave BIU then sets SENI high and latches data input from the local bus into the Data In register.
8. The slave BIU checks the validity of the data word using the EDAC unit.
9. The slave BIU generates the necessary status signals, asserts intermediate acknowledgement ($TM_0$) and places the data on the NuFTbus.
10. On the next bus clock the slave BIU deasserts $TM_0$ and tristates the data lines.
11. The slave BIU waits for n clock pulses before proceeding with the next data transfer (The number of clock cycles to wait is preprogrammed).
12. The slave BIU decrements the word count, increments the address, puts the new address on the local processor bus and goes back to step 4.
13. After the transfer of the last data word, the slave BIU asserts ACK (instead of $TM_0$) to indicate the end of block transfer.
14. With the next bus clock, the slave BIU deasserts ACK and data lines and goes back to its initial state.

### 4.6.7 Block write (master side)

The following is the sequence of steps that takes place in the master BIU when a block write is performed.

1. A negative going edge on SENO as well as on BLKEN, and a low on RWBO determines that the operation is a block write operation.
2. The BIU latches the address, data and control information into appropriate registers.
3. The BIU decodes the address and determines that it should start a NuFTbus transaction for writing the data.

4. The BIU outputs a high on WAITO.
5. The BIU generates/checks the check bits for address and data word depending upon the AEPR and DEPR inputs, respectively.
6. The BIU arbitrates for the control of the NuFTbus. Once the BIU obtains the control of the NuFTbus, it asserts the START signal on the NuFTbus and outputs the control information on the control lines, and the address on address/data bus.
7. The BIU deasserts START and the address on the next bus clock, outputs data on address/data lines, and waits for an intermediate acknowledgement.
8. With input $TM_0$ going low (as intermediate acknowledgement), the BIU deasserts data lines and latches the status information. Using the EDAC it determines the validity of the status.
9. The BIU deasserts the WAITO output to the processor.
10. 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 address and data word, goes to step 4, and continues with the next word data transfer.
11. Both SENO and BLKEN going high indicate the end of block transfer, and the BIU goes back to its initial state.

### 4.6.8 Block write (slave side)

The following is the sequence of steps that takes place in the slave BIU when a block write is performed.

1. When the START signal goes low on the NuFTbus the potential slave BIU latches the address and control information and passes it through the EDAC 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 write operation is to be performed.
3. The slave BIU requests release of the local processor bus. Upon receipt of the release acknowledge signal (RELACK), the slave BIU places the address and data on the local bus and generates the necessary control signals.
4. The slave BIU sets SENI low.
5. The slave BIU waits for two clock cycles and then checks the status of the WAITI input.
6. If WAITI is high, the slave BIU waits till WAITI goes low.
7. The slave BIU then sets SENI high.
8. The slave BIU generates the necessary status signals and asserts intermediate acknowledgement ($TM_0$).
9. On the next bus clock the slave BIU deasserts $TM_0$.
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).
11. Then 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, and puts the address and data on the local processor bus, and goes to step 4.
12. After the transfer of the last data word, the slave BIU asserts ACK (instead of $TM_0$) to indicate the end of block transfer.
13. With the next bus clock, the slave BIU deasserts ACK and the data lines and goes back to its initial state.

## 4.7 BTC-NBC interface

As mentioned earlier, the BIU has two controllers, the BTC and the NBC. Each of the read and write protocols explained in the previous section is divided into two parts. The protocol for interfacing with the processor is carried out by the BTC and that interfacing with the NuFTbus is carried out by the NBC. The NBC and the BTC interact through their internal signal handshake.

### 4.7.1 Necessity of two controllers

The read/write protocols can be implemented using a single controller (by combining two state machines the BTC and the NBC into one). This controller will implement the protocols with fewer handshake lines within the BIU but it may not be as efficient as a BIU with two controllers. The BIU is designed so that it will always decode all the addresses placed on the local processor bus. If the address is for an operation within the module,there will not be any corresponding NuFTbus transaction. Also, whenever the NuFTbus has a start cycle, the NBC will latch and decode the address in order to determine if it should respond to the transaction. Thus, the task of keeping track of transactions on the processor side as well as on the NuFTbus side of the BIU necessitates the use of two separate controllers. One controller might not be able to respond to all the transactions.

The BTC and the NBC continue decoding addresses independently until the start of an operation After that, the BTC and the NBC interact with each other using their internal signal handshake till that operation is over. Once again, they function independently. While carrying out an operation, the BTC and the NBC

- share a 32-bit EDAC
- use a common internal data path

The 32-bit EDAC is shared by means of a semaphore technique (explained in the next chapter). The large size of the EDAC in terms of gate count and area prohibits the use of two EDACs in the BIU.

The BTC and the NBC use a common internal data bus. This reduces the area, which otherwise would be used for routing 40 additional address/data lines in the BIU.

The BTC and the NBC are designed at the register transfer level. The register transfer level code is implemented in the VHDL. The VHDL code for the NBC and the BTC written by the author is documented in the reference [20]. The different tasks of the BTC and the NBC are as follows:

1. BTC is used
    a. to decode addresses from the board
    b. to generate/check EDAC on the address and data
    c. to indicate double errors in the address or data to the processor and to set corresponding flags in the BIU status register
    d. to receive the data word from the NBC and pass it to the processor
    e. to complete the BIU register read/write operation in master mode
    f. to put the master mode transaction on hold, if a slave mode transaction is already started by the NBC
2. NBC is used
    a. to arbitrate for the NuFTbus

b. to start a NuFTbus transaction for read/write

c. to generate EDAC on control lines

d. to receive data from a slave on the NuFTbus, check EDAC and pass the data to the BTC

e. to monitor the NuFTbus for the bus status (busy or available)

f. to decode the addresses on the NuFTbus and respond if the BIU is being addressed

g. to put the on-board processor in the hold state by raising REL to complete a slave mode transaction

h. to retry a failed operation

i. to monitor arbitration error, retry failed inputs, and take necessary action

j. to decrement block size and increment block address in block transfer

k. to wait for a specified number of cycles between transfers in the block mode

l. to set the error flags in the BIU status register and to inform the processor about the error

m. to complete a BIU read/write operation in the slave mode

### 4.7.2 NBC-BTC handshake

Whenever the BTC decodes an address and identifies that it should start a NuFTbus transaction, it sends a signal to the NBC indicating the start of a master mode transaction. The NBC responds to that signal with an acknowledgement and then the operation is carried out.

Whenever the NBC decodes an address and identifies that it should start a NuFTbus transaction, it sends a signal to the BTC indicating the start of a slave mode transaction.

The BTC responds to that signal with an acknowledgement and then the operation is carried out. This handshake is accomplished by defining internal request and acknowledgement signals between the BTC and the NBC.

**Deadlock Problem:** Consider a situation when a module starts a NuFTbus transaction (it wants to be a bus master), but it is also a potential slave for a NuFTbus transaction already started on the NuFTbus. In this case, the BTC would latch the address from the module in the Address Out Register and put the processor in the wait state. The NBC would latch the address from the NuFTbus in the Address In Register and request the release of the local bus. Then, both the NBC and the BTC would wait for acknowledgement from each other and this would lead to a deadlock (deadly embrace) situation.

To avoid this situation, the NBC has been given priority over the BTC. (In other words, a transaction on the NuFTbus has higher priority). Accordingly, if both the NBC and the BTC request each other, then the BTC would generate acknowledgement and proceed with the transaction on the NuFTbus (making the BIU a slave). Once that transaction is over, the slave BIU would become the master and complete the transaction that is on hold. If the BTC is given priority over the NBC, the NBC would have to send a 'Try again later' acknowledgement on the NuFTbus. This would lead to an error signal from the master BIU to the master processor. The processor would then have to roll back within the instruction queue and try to execute the transaction again. This would not be an easy task. On the other hand, when priority is given to the NBC, none of the processors are involved, and the deadlock situation is resolved by the BTC and the NBC.

The handshake signals between the BTC and the NBC are defined as follows:

1. TRANSFER OUT READY (TOR): TOR is driven by the BTC. In the master mode a high on TOR tells the NBC that it can the start arbitration followed by a corresponding NuFTbus transaction. In the slave mode, a high on TOR tells the NBC that the operation has been completed by the BTC, and the NBC can send back an acknowledgement.

2. READY FOR DATA (RFD): RFD is driven by the BTC. When RFD is high, it indicates that the BTC is waiting for a data input from the NBC.

3. TRANSFER IN PROGRESS (TIP): TIP is driven by the NBC. A high on TIP indicates that the NBC is currently busy 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 the NBC receives RELACK from the processor.

4. ADDRESS IN READY (AIR): AIR is driven by the NBC. In the slave mode, AIR going high indicates that the NBC has a valid address ready from the NuFTbus for a read and write transaction. In the master mode, AIR going low indicates to the NBC that the transaction was completed.

5. DATA IN READY (DIR): DIR is driven by the NBC. It is used in the slave write mode when the data from the NuFTbus is available for the BTC.

6. READ/WRITE NUFTBUS OUT (RWNO): It is driven by the BTC. A high on RWNO indicates a master read operation and a low indicates a master write operation.

7. READ/WRITE NUFTBUS IN (RWNI) : It is driven by the NBC. A high on RWNI indicates a slave read operation and a low indicates a slave write operation.

8. NBC LOCK: It is driven by the BTC. A high indicates that the BTC is ready to pass a master operation to the NBC.

9. NBC ACK: It is driven by the NBC in response to NBC LOCK.

10. BTC LOCK: It is driven by the NBC. A high indicates that NBC is ready to pass a slave operation to the BTC.

11. BTC ACK: It is driven by the BTC in response to BTC LOCK.

12. SBLKRD: It is driven by the NBC. A high indicates that the slave operation is a block read operation.

13. SBLKWR: It is driven by the NBC. A high indicates that the slave operation is a block write operation.

All of these signals have similar functions during block operations.

The NBC and the BTC also control different small logic blocks which carry out operations such as:

- incrementing the address in the block mode
- decrementing the word count in the block mode
- keeping track of the bus status (busy or available)
- providing programmable delay between two transfers in the block mode
- counting the number of retries for a failed operation
- counting the number of transactions in the bus lock mode

These function blocks along with their circuits are explained in the next chapter.

# 5.0 NUFTBUS BIU DESIGN

## 5.1 Introduction

This chapter explains the operation of the address decode unit, the arbitration unit, error detection and correction units and other logic blocks performing functions such as counting the number of retries, incrementing the block address, bus monitoring and bus lock detection.

## 5.2 Address Decode Unit

Operations on the NuFTbus are not synchronized to the operations on the board. So the BIU may be addressed as the master BIU and as the slave BIU at the same time. Both addresses have to be decoded in order to determine if the BIU has to take part in that particular transaction. Thus, two address decoding units are necessary. One is associated with the BTC, and the other one is associated with the NBC. (Both are identical in function).

The inputs to the address decode units are:

- board ID
- start address of on-board global memory
- size of the on-board global memory (n)
- BIU register array addresses
- incoming address

The size of the on-board global memory is given as a 5-bit binary number n. The number n implies that there are $2^n$ words ($2^{n+2}$ bytes) of on-board global memory. Therefore, $0 \leq n \leq 30$. It is loaded in the BIU register at the time of initialization. The on-board global memory is contiguous. The start address is also loaded during initialization. The ID code is needed to determine the board's address in slotspace. (Slotspace for a board with ID of $S_i$ is F$S_i$XXXXXXH).

There are three output signals. They are:

- SLOTSPACE: SLOTSPACE goes high to indicate that the incoming address is in the slotspace of the board.
- BIUSPACE: BIUSPACE goes high to indicate that the incoming address is for one of the BIU registers.
- GLOBAL: GLOBAL goes high to indicate that the incoming address is in the global space on the board.

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

Both the BTC and the NBC interpret the outputs from address decode units in different ways. For an address from the NuFTbus, the SLOTSPACE/GLOBAL signal is being addressed as a slave. So, the NBC starts a corresponding slave read/write operation. For an address from the board, the SLOTSPACE/GLOBAL signal going high tells the BTC that the operation would be completed on the board itself. In such a case, the BTC monitors the SENO line till the operation is complete but does not take part in the operation.

Two SLOTSPACE 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 need not have the same decoding logic which is already present in the BIU. The board does the decoding of only the lower address lines in each range. 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.

### 5.2.1 Address detection mechanism

Each board has a slotspace which ranges from F$S_i$000000H to F$S_i$FFFFFFH, where $S_i$ is the 4-bit ID code. Address lines $A_{31}$-$A_{24}$ of the incoming address are compared with F$S_i$H. If they match, the output signal SLOTSPACE goes high. Global address space is detected by masking the lower address lines. For the 5-bit memory size value n, the lower n+2 address lines are masked (set to 0) and then the modified incoming address is compared with the start address. If they match, the output signal GLOBAL goes high.

The actual circuit for address decoding operates as follows. A 5 line to 32 line decoder decodes the 5-bit memory size signals. Corresponding to the number n, the output $Y_n$ of the decoder goes low. Depending on $Y_n$ , a masking pattern $X_{31}$-$X_2$ is created in which $X_{31}$-$X_{n+2}$ are ones and $X_{n+1}$-$X_0$ are zeros. The incoming address is ANDed with the masking pattern to produce an 'effective' starting address. Then the address is compared with the memory start address. If they match, the incoming address is in the global memory on-board. As n is fixed and the starting address is fixed during initialization, the only delay in the address decoding scheme is in masking and comparison operations, which is equivalent to 5 standard gate delays. The address decoding scheme is shown in Figure 6 on page 66 and in Figure 7.

## 5.2.2 Examples of address detection mechanism

Let the ID for a board be 4. Then its slotspace is F4XXXXXXH. Address lines $A_{31}$-$A_{24}$ of the incoming address are compared with F4H. Thus any address between F4000000H and F4FFFFFFH is detected as the address in the slotspace and correspondingly the output SLOTSPACE signal goes high.

Let n for this board be 14. Then the on-board global memory has $2^{14}$ words or $2^{16}$ bytes. Let the start address of the on-board global memory be 9A160000H (Note that the last n+2 bits of the start address will always be zeros). Thus, the global memory range is 91A6XXXXH. Let an incoming address be 328CA79AH. The lower n+2 address lines are masked by decode logic. So 328C0000H is compared with 9A160000H. Since they

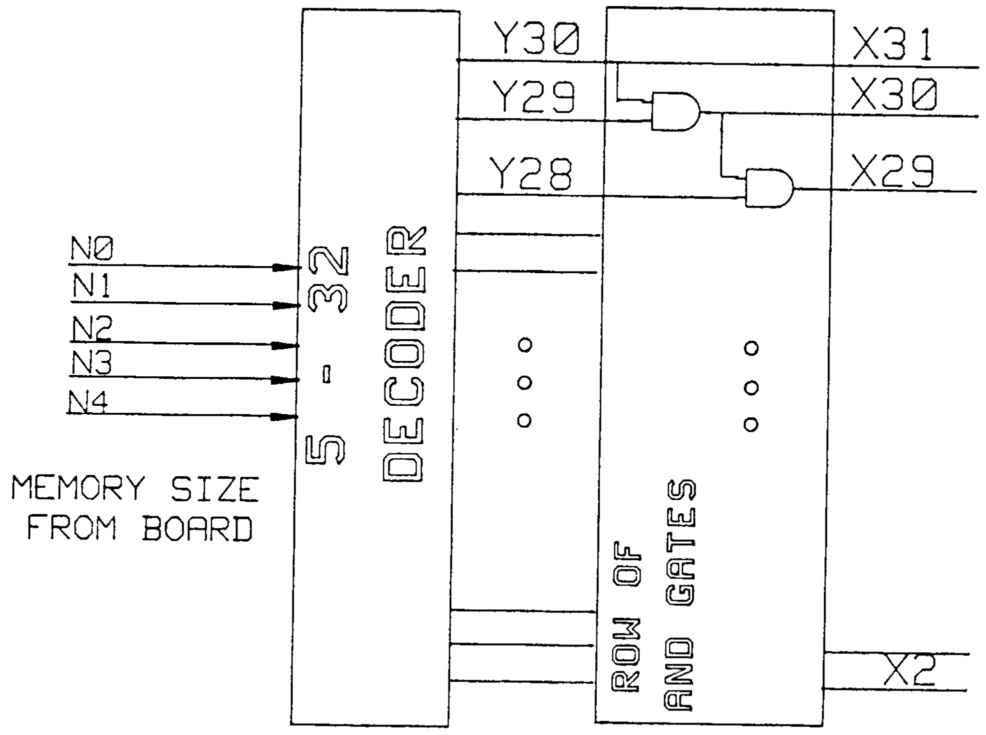


Figure 6. Address decode logic

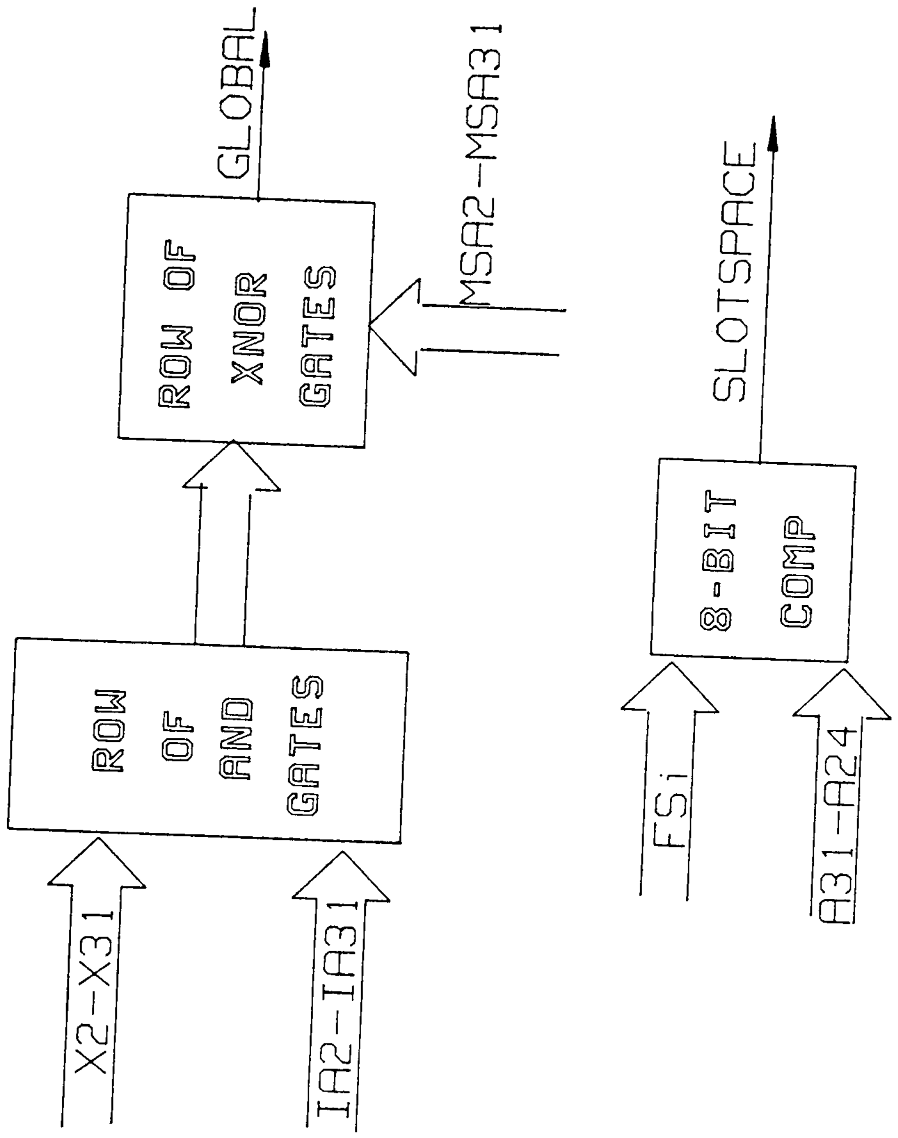


Figure 7. Address decode logic (continued)

do not match, the incoming address is not in the global address space of the board. Consider another example. Let the incoming address be 9A162153H. After masking lower 16 lines, incoming address matches the start address. So, the output GLOBAL signal goes high to indicate that the address is in the global memory space.

Consider an example with n equal to 16. Then the on-board global memory has $2^{16}$ words. Let the start address of the on-board global memory be 10180000H. Let an incoming address be 101B5498H. The lower n+2 address lines are masked. So, 101B5498 changes to 10180000H. It matches the start address and the global output goes high.

## 5.3 Arbitration unit

The arbitration operation with one bit error detection is explained below in detail.

The two sets of arbitration lines are named $ARB_9-ARB_7-ARB_5-ARB_3-ARB_1$ and $ARB_8-ARB_6-ARB_4-ARB_2-ARB_0$. Modules, while competing for the bus, place the complement of each bit of their ID code on each of the two sets of arbitration lines. Let the ID code be represented as $ID_4-ID_0$. $ID_4$ (msb) is placed on lines $ARB_9$ and $ARB_8$, while $ID_0$ (lsb) is placed on lines $ARB_1$ and $ARB_0$. The arbitration operation starts from the msb. If a module wins the arbitration contest at the msb (i.e., $ID_4$ is accepted at both $ARB_9$ and $ARB_8$ lines), arbitration proceeds to the next ID code bit. If the results differ at an ID bit, that bit is bypassed and further arbitration is carried out at lower bit levels. But if there is an error in the arbitration unit (the ID bit is not accepted at two locations), then the error flag (ARBERR) is raised. Subsequently, that module removes itself from the bus arbitration scheme.

Bit errors are modeled as one of the two types, either stuck at 1 (s-a-1) or stuck at 0 (s-a-0). As an example of the operation of the arbitration unit, assume that $ARB_9$ line is s-a-1. Let a module X have an ID code 01010. While competing for the bus, X places a 0 on the $ARB_9$ and $ARB_8$ lines. $ARB_9$ is s-a-1 and remains high while $ARB_8$ is pulled low. When the two lines do not match, the arbitration unit of module X bypasses $ID_4$ and arbitration proceeds with the remaining four ID bits. This is also true for all other modules, and thus $ID_4$ is ignored by all modules. Now the module with the lowest 'modified' ID code, $ID_3 - ID_0$ (4 bits) wins the arbitration contest. The same will be true if $ARB_8$ is s-a-1.

Now let $ARB_7$ be s-a-1. All of the modules compete at the $ID_4$ bit level. Those modules with $ID_4 = 0$ continue arbitrating at the next lower level. With $ARB_7$ s-a-1, $ID_3$ will be bypassed and the module with the lowest 'modified' ID code, $ID_4 - ID_2 - ID_1 - ID_0$ will win the arbitration. The arbitration proceeds in the same way for the rest of the arbitration lines which are s-a-1.

Now consider a s-a-0 condition. When any ID bit = 0 for a module it always has priority over the corresponding ID bit = 1 of other modules, as the ARB lines are open collector. (The ARB lines are WIRE-ANDed together). Let $ARB_9$ be s-a-0. Let module X (ID code 01010) place a 0 and module Y (10111) place a 1 on lines $ARB_9$ and $ARB_8$ . The 0 of module X pulls down the $ARB_8$ line and $ARB_9$ is already low since it is s-a-0. So module X wins the arbitration at $ID_4$ and proceeds to arbitrate at the lower order bits. Module Y sees a 0 on both lines instead of its 1 and terminates its own arbitration. Thus the bit error has no effect on the arbitration in this case. If other competing modules have $ID_4 = 1$, then they will likewise terminate arbitration after checking the msb level.

Any module which has $ID_4 = 0$ will continue to arbitrate at the lower bit level along with module X.

The first ID code bit, $ID_0$, is the parity bit. So all ID codes have a distance of two between them. With a single bit error, the 'modified' ID codes always have a distance of at least one between them.

For example, the ID code of module 8 is 10001 and the ID code of module 9 is 10010. ($ID_0$ = parity bit; even parity). With $ID_3$ s-a-1, the 'modified' ID codes are 1001 and 1010. Modules 8 and 9 still have a distance of two between them. With $ID_1$ s-a-1, 'modified' ID codes are 1001 and 1000. Now modules 8 and 9 have a distance of one between them. Still, according to the arbitration scheme explained above, module 8 will get the control of the bus. 'Modified' ID codes of any two modules can never be the same (distance of zero). Hence no two modules can win the arbitration at the same time.

Thus, in the event of either s-a-1 or s-a-0 faults, arbitration will be carried out successfully. In all cases of single bit errors, one module will win the arbitration and no two modules will win the arbitration at the same time. This provides the desired fault tolerance effect.

It is clear from the above discussion that the arbitration is not dependent on the number of modules arbitrating at a time. When any one of the modules places a 0 on the ARB line, the line is pulled down irrespectively of the other modules placing a 0 or a 1 on that line. Also, in the case when all the modules which are arbitrating place a 1 on the ARB line, the ARB line goes high. The module with the lowest ID code or the lowest 'modified' ID code (in case of one bit errors) will always get the control of the bus.

To test the arbitration scheme and determine the timing constraints of the system, simulations are performed on a system containing four modules. As explained above, this is valid for any number of modules (as long as all the combinations of placing 1's and 0's on the ARB lines are considered) because of the basic properties of the system. The simulation results are presented in the next chapter.

### 5.3.1 Design of the Arbitration Unit

The NuFTbus arbitration mechanism is based on a priority arbitration scheme (the module with the lowest ID code wins control of the bus), which is different from those schemes that have strict priority arbitration. The Nubus arbitration scheme distributes the bus bandwidth evenly among all the modules and is thus referred to as a fair priority arbitration.

Each module which needs control of the bus asserts (pulls down) the RQST line. (RQST is open collector). Also, RQST can only be asserted when it is in the deasserted (high) state. Once RQST is asserted by a particular module, arbitration begins at the next rising edge of the bus clock. All the modules which request the bus before this edge can contend for the bus. All other modules must wait (even if they are of higher priority) until every module which requested the bus in a previous period gets control of the bus and completes its respective transaction.

When a module completes a transaction, it releases the RQST line (if it does not want to lock the bus). Then the rest of the modules with RQST asserted compete again to obtain control of the bus. When the last module releases the RQST line, RQST goes high (as no module is pulling it low). With RQST high, the modules which could not participate in the previous round of arbitration, pull RQST low and start the arbitration. Thus, in this scheme a module with a lower priority gets a fair chance to get the control of the bus.

The arbitration circuitry consists of three subcircuits and a counter [19].

- Subcircuit RQFF: Whenever a module wants the control of the bus it sets an Internal Request line INTRQ. INTRQ is reset as soon as the module gets the bus. If RQST is high and INTRQ goes high, then with the next rising clock pulse of the bus clock the module pulls RQST low. RQFF also generates one bus clock wide pulse to start the arbitration. If the RQST line is already low, INTRQ is held high and the module waits for the previous bus master's operations to complete. The circuit diagram for RQFF is shown in Figure 8.

- Subcircuit CGIA: CGIA places one bit of the ID code of the module on two arbitration lines of the bus (one in each set). The output of CGIA is open collector. Thus, after placing a high on the line, if the line is still low, the CGIA realizes that some other module has placed a zero on the line and the CGIA deasserts its own grant signal (since zero has priority over one in arbitration). Further arbitration by this module is disabled. If the CGIA places a 0 on the bus, it pulls the ARB line low and continues with the arbitration at lower order bits. The circuit diagram for CGIA is shown in Figure 9. Simulation results are discussed in the next section.

  The operation of the CGIA can be briefly explained by the following pseudo code.

```
    if compete = true then
```

```
        arbbus = id'
*   The arbbus line is WIRE-ORed with
*     arbbus lines of other modules.
        if arbbus = id' then
   grant = true
    else
   grant = false
    endif
     else
    arbbus = 1
    grant = false
     endif
```

- Subcircuit ABARB: The fault tolerant NuFTBus has two sets of arbitration lines. Arbitration is carried out on both sets simultaneously. ABARB compares the two arbitration lines corresponding to a single ID line. If 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 proceed. 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. The subcircuit ABARB is shown in Figure 10. The pseudo code for the ABARB circuit is shown below.

```
 if ainb = binb  then
aout = ainb
bout = binb
arb-err = false
 else
if aing = bing then
aout = aing
bout = bing
arb-err = false
else
aout = 0
bout = 0
arb-err = true
endif
 endif
```

where

ainb, binb: grant signals from the two CGIA subcircuits for the ID code bit i

aing, bing: grant signals from the two CGIA subcircuits for the ID code bit i-1

aout, bout: outputs going to the two CGIA subcircuits for the ID code bit i+1

arberr: error signal to flag an error in the input lines

The schematic of a subcircuit ARBI, consisting of the CGIA and the ABARB is shown in Figure 11.

Arbitration logic is simulated using standard LS TTL delays (propagation delay of 10-15 ns/gate). According to the specifications of the NuFTbus, the bus contention is to be settled in two clock cycles. With the LS TTL delays, the arbitration lasts for four clock cycles. By implementing the circuit on a Gate Array, with reduced propagation delays of the order of 1-3 ns/gate, the arbitration would be completed in two clock cycles.

Each module has two grant lines $G_{0X}$ and $G_{1X}$ for module X. A high on both $G_{0X}$ and $G_{1X}$ indicates that module X has won the arbitration. To synchronize the arbitration operations with the bus clock, a counter is used. Whenever RQST goes low (start of a new round of arbitration) or when there is an ACK pulse on the bus (end of one of the intermediate transactions), the counter starts. When the count reaches four, the grant lines $G_{0X}$ and $G_{1X}$ are latched. At the next clock pulse, the module pulls down its compete lines $COMP_0$ and $COMP_1$ to stop the arbitration. As soon as a module obtains the control of the bus, the arbitration unit pulls down the INTRQ line. At the end of its transaction, the signal SET reinitializes the request flip flop and waits for the next INTRQ signal to start another arbitration. The schematic for the arbitration unit for a module is shown in Figure 12.

## 5.4 Error detection and correction unit

There are two error detection and correction (EDAC) units in the BIU. One is a 32-bit EDAC for address/data path (addr/data EDAC) and the other one is a 4-bit EDAC for control lines $TM_1$ , $TM_0$, START and ACK (control EDAC). Control EDAC is used exclusively by the NBC while addr/data EDAC is shared by both the BTC and the NBC. The EDAC unit is a combinational circuit followed by an output register.

The control signals interfacing with the EDAC are

1. GEN_CHK (input): When GEN_CHK is high, the EDAC is in the generate mode. It generates the check bits and the parity for the incoming data. When GEN_CHK is low, the EDAC is in the check mode. It checks for single and double errors in data.
2. DOUBLE ERROR (output): DOUBLE ERROR going high indicates that there is an error in the data being checked and the error is at two or more bit positions. ( If a double error is identified, then the BIU retries the operation (in master mode) or sends an error acknowledgement (in slave mode)).
3. EDACRI (input): A rising edge on EDACRI latches the correct data, check bits and parity in the output register.
4. EDACOE (input): A high on EDACOE enables the register output. The outputs are tristated when EDACOE is low.

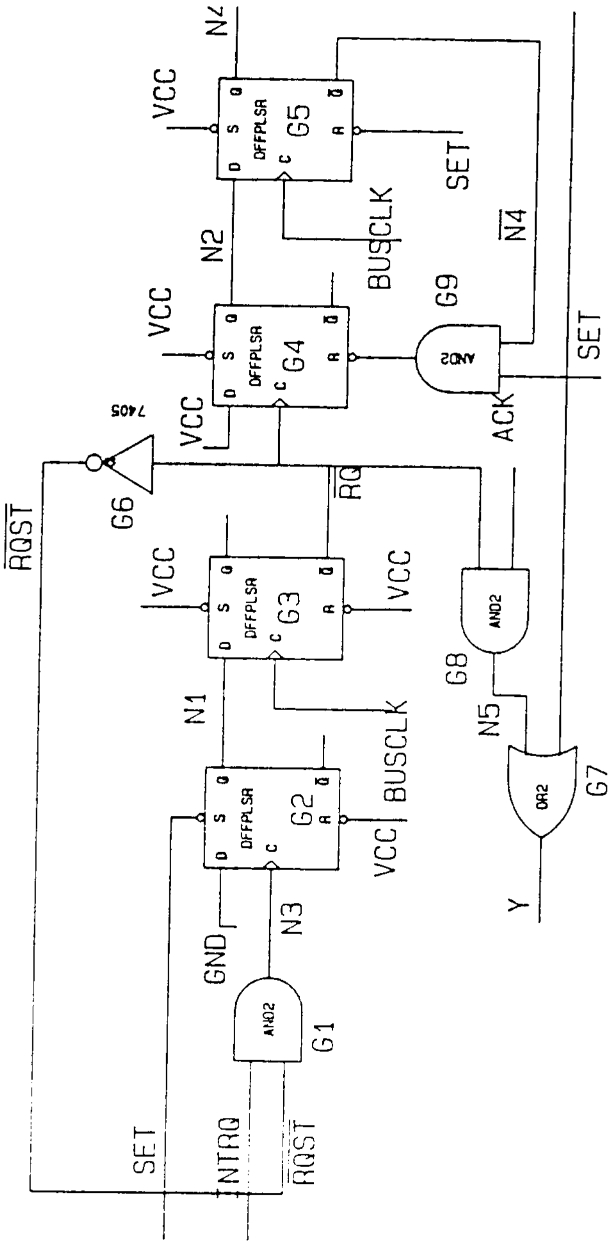


Figure 8. Subcircuit RQFF.

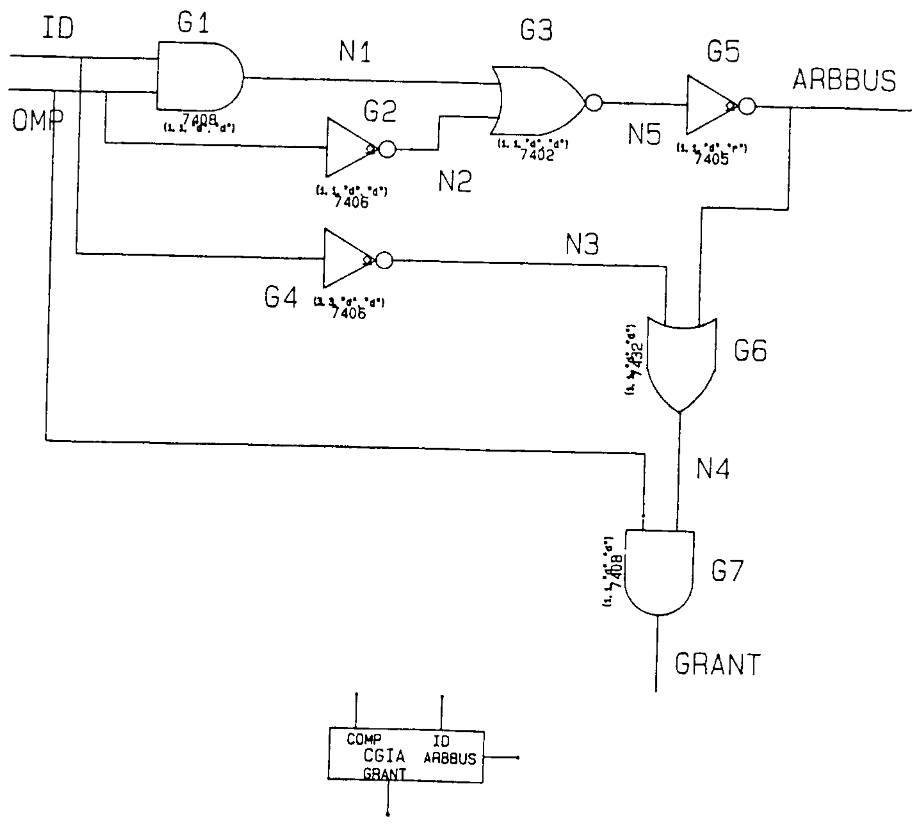


Figure 9. Subcircuit CGIA.

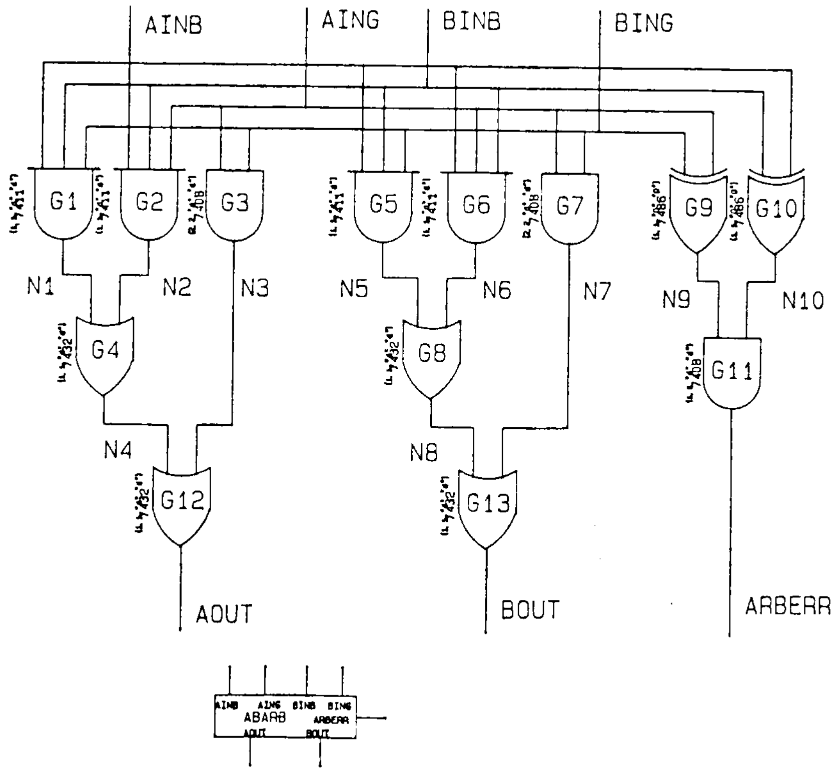


Figure 10. Subcircuit ABARB.

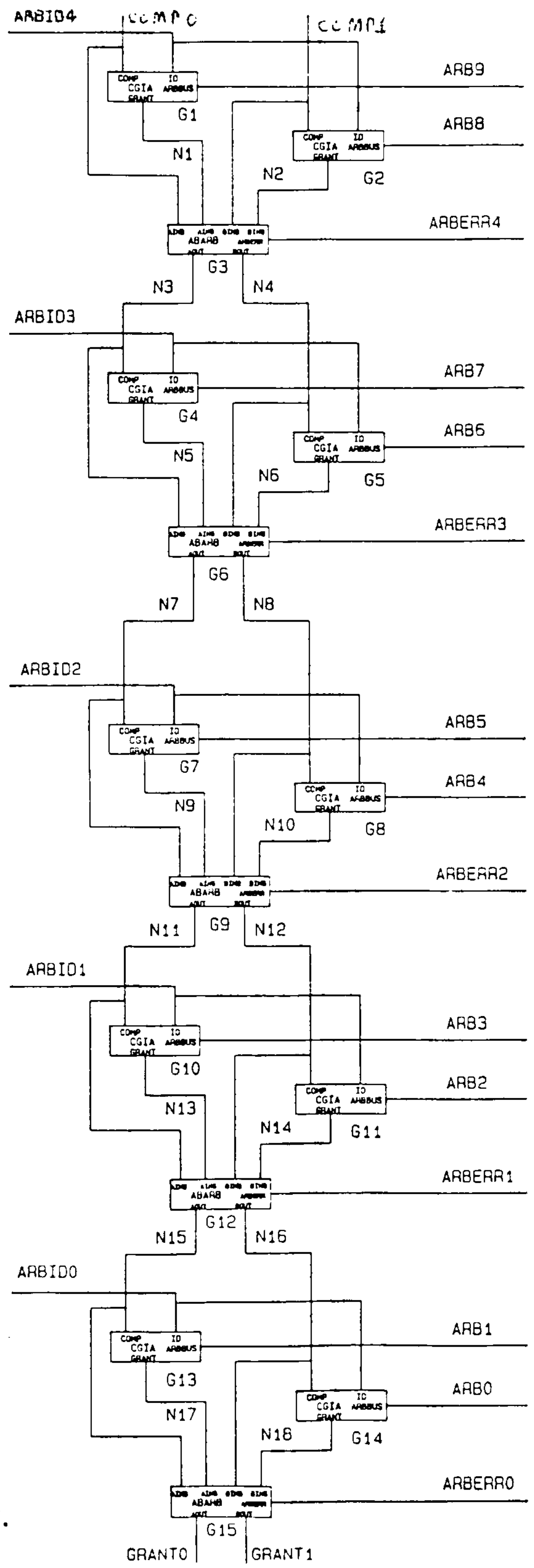


Figure 11. Subcircuit ARBI.

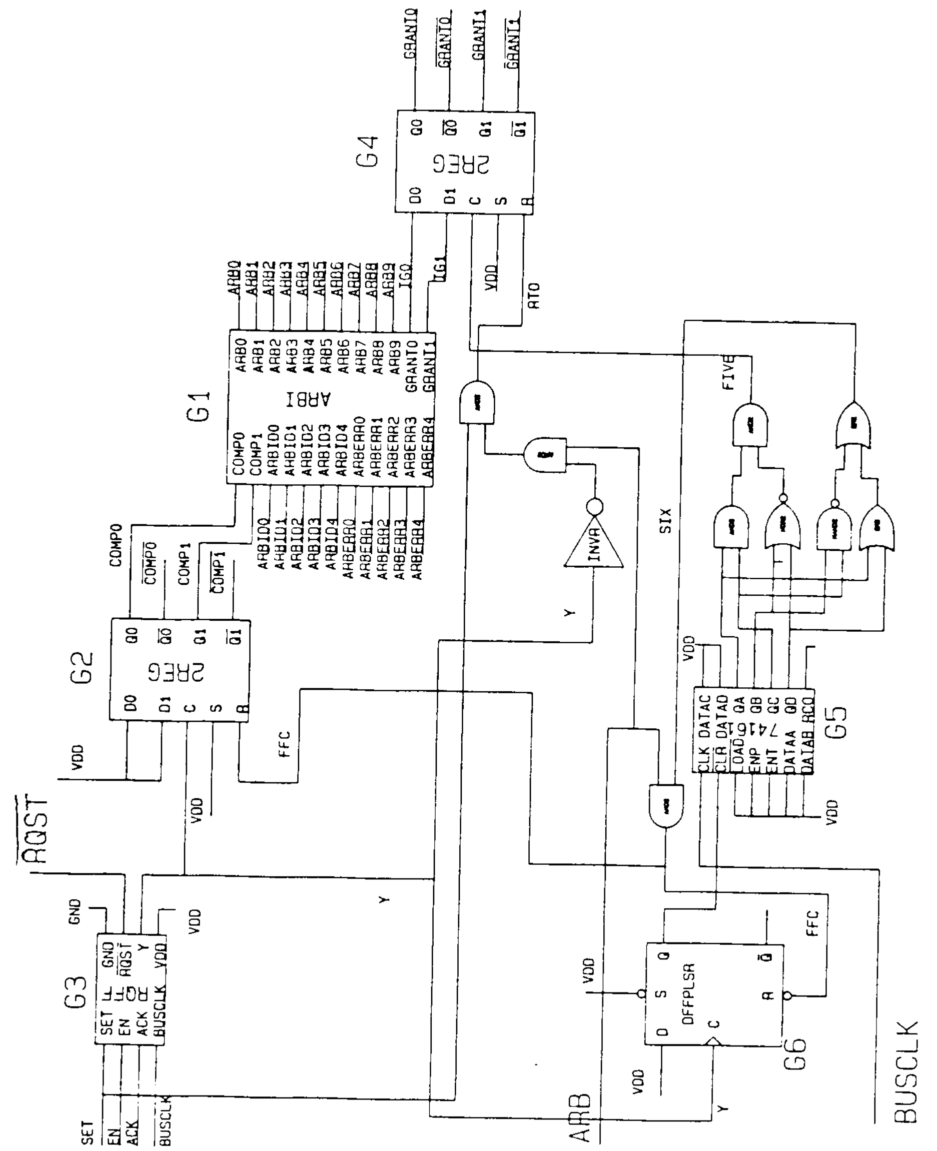


Figure 12. The Arbitration Unit - ARBITER.

The control EDAC takes four control lines as input and produces three check bits and a parity bit, while the addr/data EDAC takes 32-bit word as input and produces seven check bits and a parity bit.

### 5.4.1 EDAC sharing

The NBC and the BTC share the addr/data EDAC using special handshake signals. This is shown in Figure 13. Whenever the NBC needs the EDAC, it checks the state of EDACBUSY2 line. If it is low then the NBC sets ACQEDAC2 line and gets control of EDAC. Whenever the BTC needs the EDAC, it sets 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. Consider the different possible cases to understand the handshaking.

- EDAC available: The BTC (NBC) checks the EDACBUSY1 (EDACBUSY2) line. If it is low, the BTC (NBC) sets the ACQEDAC1 (ACQEDAC2) line and gets control of the EDAC.
- EDAC busy: The BTC (NBC) waits till the EDAC becomes available and then gets control of it by setting the ACQEDAC1(ACQEDAC2) line.
- Both the BTC and the NBC need the EDAC: A problem would arise if the EDAC was available and both the BTC and the NBC checked the ACQEDAC lines at the same time. Both would get a low on the EDACBUSY lines, and so both would assume that they could get the EDAC. This is avoided by adding the CHKEDAC signal from the BTC. The BTC sets CHKEDAC, one cycle before it checks the EDACBUSY1 line. By doing this, 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 is available at that time. the BTC will get control of the EDAC and once its operation is complete, the NBC can get the EDAC. Thus the addition of the CHKEDAC line solves the problem of both the BTC and the NBC trying to access the EDAC at the same time.

The following sections describe the operation of and hardware for functional blocks used by the NBC and the BTC.

## 5.5 Bus lock detect mechanism

The NuFTbus specifications allow the master to lock the bus. The master would lock the bus for operations such as Test and Set of semaphores and sharing common resources. The maximum time for which the bus can be locked is not specified in the NuFTbus specifications. It is to be decided in the system specifications. Each system will set a limit on the maximum number of transactions allowed in the lock mode. A 4-bit register is provided in the BIU which is loaded with this number during initialization.

All the BIUs on the NuFTbus keep track of the transactions on the bus. When a BIU is contesting for the bus but does not win the bus, it latches the ID of the winner module and checks if the winner module is locking the bus. If the winner module tries to lock the bus for more than the specified number of times, the BIU generates an attention cycle and restarts the arbitration process. The circuit for this procedure is shown in Figure 14.

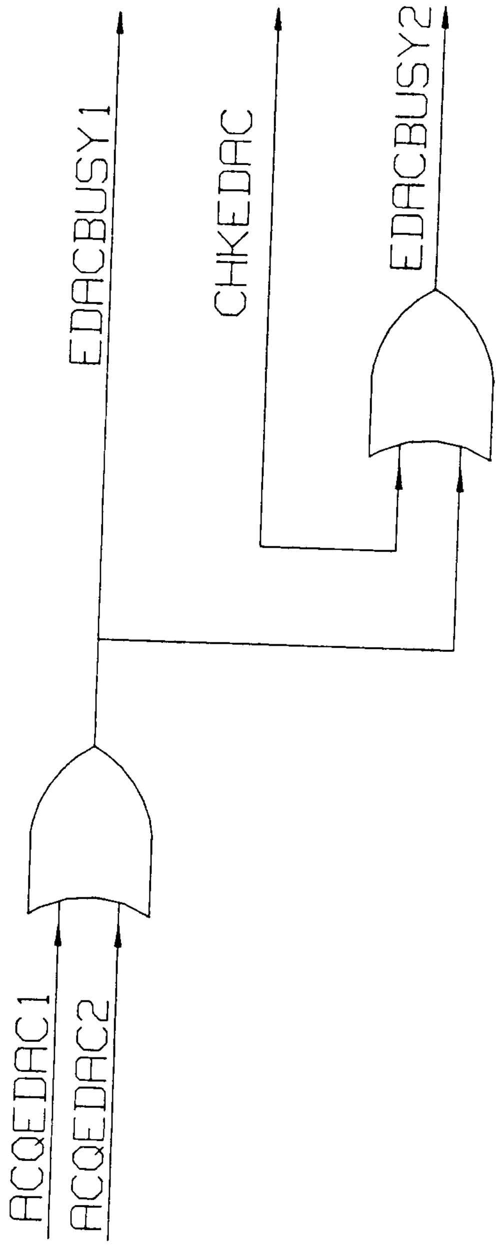


**Figure 13. Interface between NBC and BTC for EDAC sharing**

**Operation:** Initially, both latches are reset and the output of the comparator is low. (It goes high when the current winner ID and the last winner ID are different). Also the counter is reset. Falling edge of START indicates the start of a new transaction. The ARB lines have the ID of the winner module. This ID is stored in latch A. It is compared with the last winner ID which is already stored in latch B. (For the very first transaction this would be 0). If the two IDs are different, then the current winner ID is stored in the latch B and the counter is reset. This ID is compared with the winner ID of the next transaction and so on. If the ID for two consecutive operations is the same, due to bus locking, the counter is incremented at the rising edge of START.

If the count matches the maximum number of times a bus can be locked (stored in a register), a RSTBUS (restart bus) signal is given to the NBC. Then the NBC generates an attention cycle (START and ACK both low) for rearbitration. Generation of the attention cycle also resets the counter.

## 5.6 Bus status detect

Once a module wins the arbitration, it can start a NuFTbus cycle immediately if the bus is available, otherwise it has to wait till the previous module finishes its operation. The BIU keeps track of the bus status, whether the bus is busy or available. This is done by the circuit shown in Figure 15. A low on START and a high on ACK starts a NuFTbus transaction, while a low on ACK ends it. So, with a low on START and a high on ACK, the flip flop is set and NBC gets a high on the BUSBUSY signal. ACK going low clears the flip flop and BUSBUSY goes low.

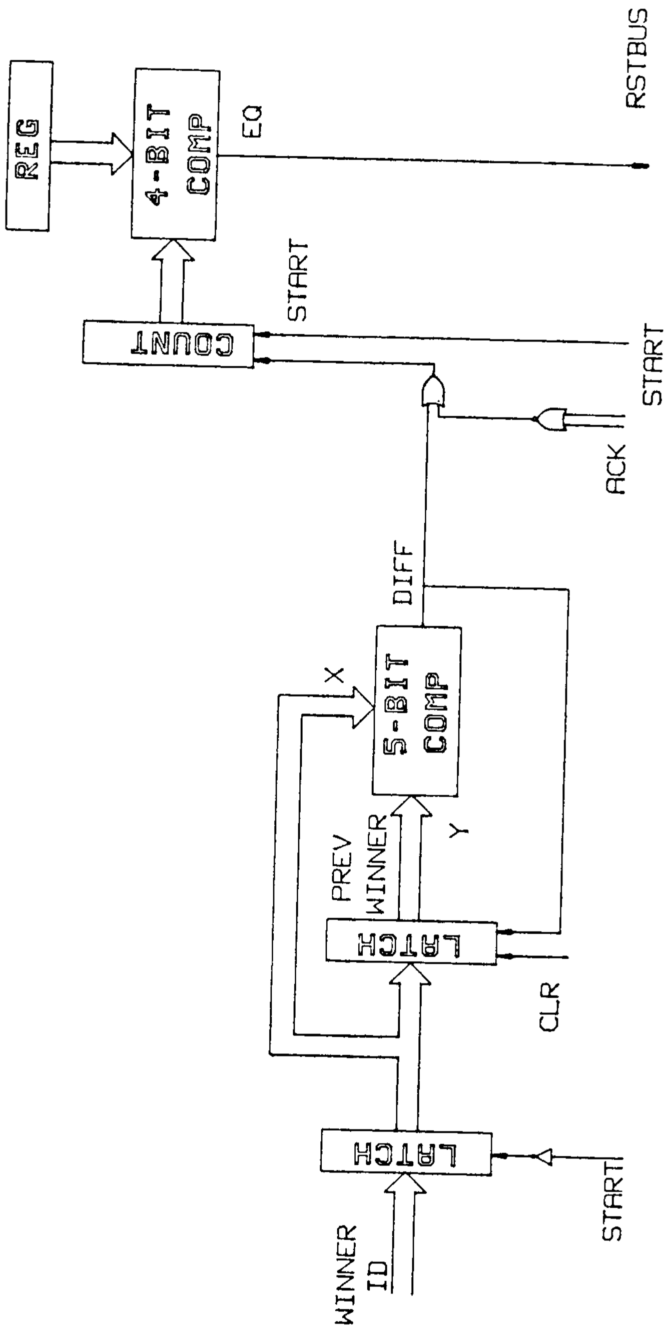


Figure 14. Detection of bus locking

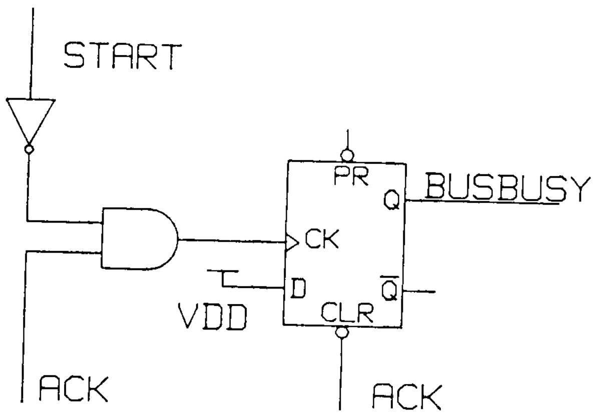


Figure 15. Bus status detect

## 5.7 Block operations

Following are the different circuits used for block operations. A block operation is detected by a high on $AD_0$ and $TM_0$ and a low on $AD_1$ lines during the START cycle.

### 5.7.1 Block size and start address

In a block operation, address lines $AD_5 - AD_2$ give information regarding block size and block start address. Figure 16 on page 88, and Figure 17 show circuits to decode the block start address and block size, respectively. Both, start address and block size, are stored in registers.

### 5.7.2 Block address incrementer

According to the NuFTbus specifications, the intermediate addresses in a block operation are not sent on the bus from the master. It is the responsibility of the slave device to increment the block address and to receive a data word in a block write operation or to send a data word in a block read operation. The increment operation is performed by the NBC in the BIU. The circuit shown in Figure 18is cascaded to make a 32-bit incrementer. The incrementer accepts a new address when INCRLAT goes high. If INCRLAT is low, the previous address is fed back to the incrementer input. The address is incremented by an INCR pulse from the NBC (INCRLAT being still high). As the output of the incrementer is connected to the addr/data bus of the BIU, the outputs are made tristatable. The outputs can be enabled by the INCROE signal. The sequence of the operation is also shown in terms of a timing diagram in Figure 18.

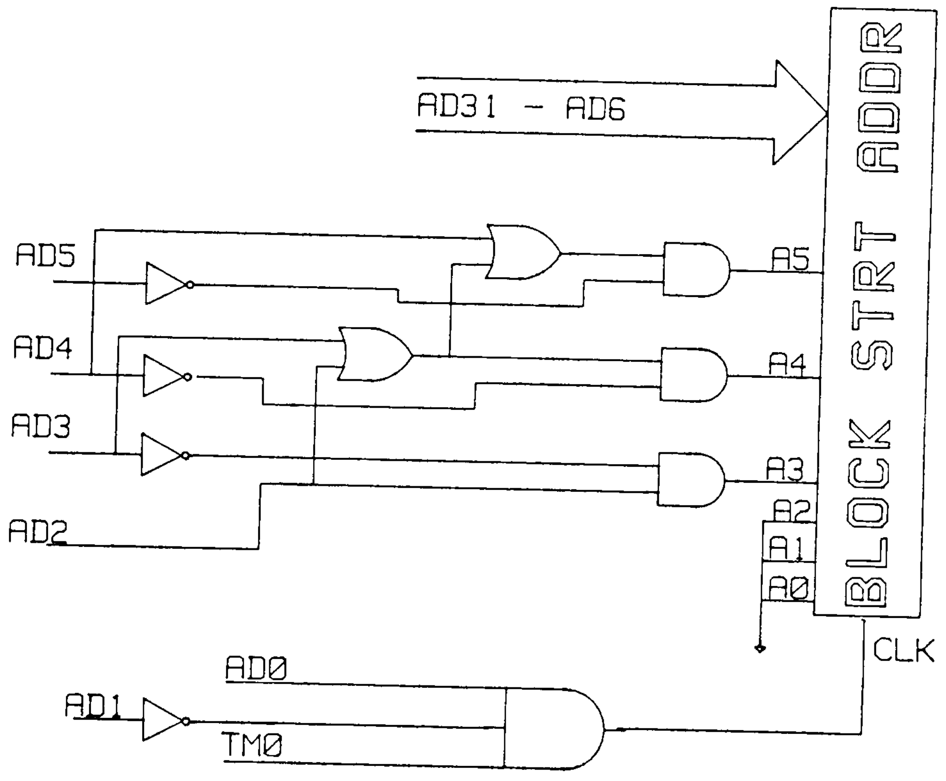


Figure 16. Block start address detect

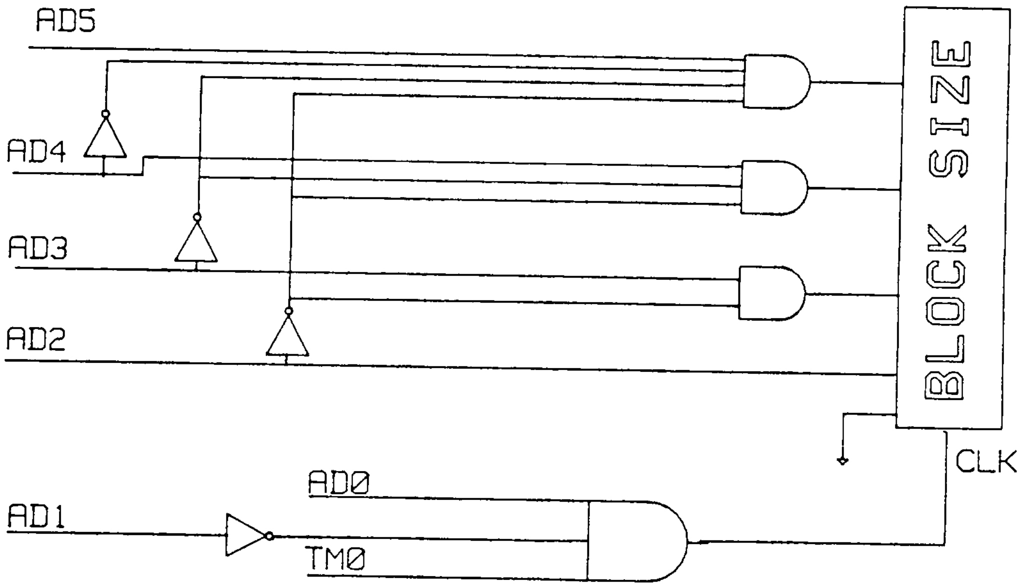


Figure 17. Block size detect

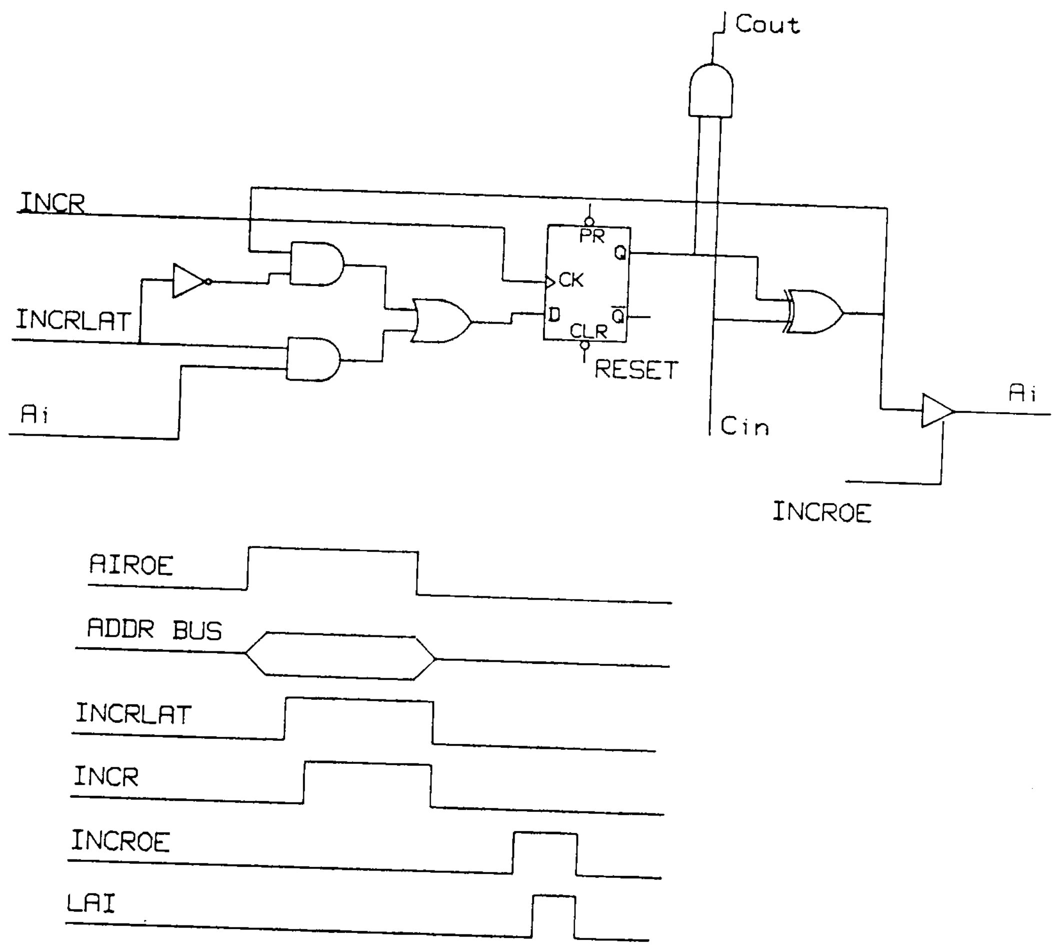


Figure 18. Block address incrementer

### 5.7.3 Block size decrementer

Block size is decremented every time an intermediate data cycle is completed. The decrementer circuit is shown in Figure 19. For the last data transfer in block mode, a signal LAST is generated which is used by the NBC in terminating the block operation after that cycle.

### 5.7.4 Wait between intermediate data transfers

The Nubus does not provide a signal which would indicate that the intermediate data transfer is done and that the next transfer should be started. In a block read operation, the slave BIU sends an intermediate acknowledgement along with the data. The master BIU sends the data to the master processor, and when it is done, it just waits for the next data from the slave. Also in case of block write operation, once the master BIU gets an intermediate acknowledgement from the slave BIU, it gets the next data from the board. It sends the data on the NuFTbus without any handshake signal to the slave. So if the operations of the master and the slave are not synchronized, some of the data might be lost/corrupted or the master might wait indefinitely for the data. To get around this problem, the slave NBC waits for a certain time interval before starting the next transfer. During initialization, a number is loaded in the BIU command register indicating the clock cycles to wait between two intermediate data transfers. This number is calculated in such a way that after that time interval, new data would be available on the NuFTbus (in case of a block write operation) or the master BIU would have finished sending the last data to the master processor (in case of a block read operation). This number is programmable and system dependent.

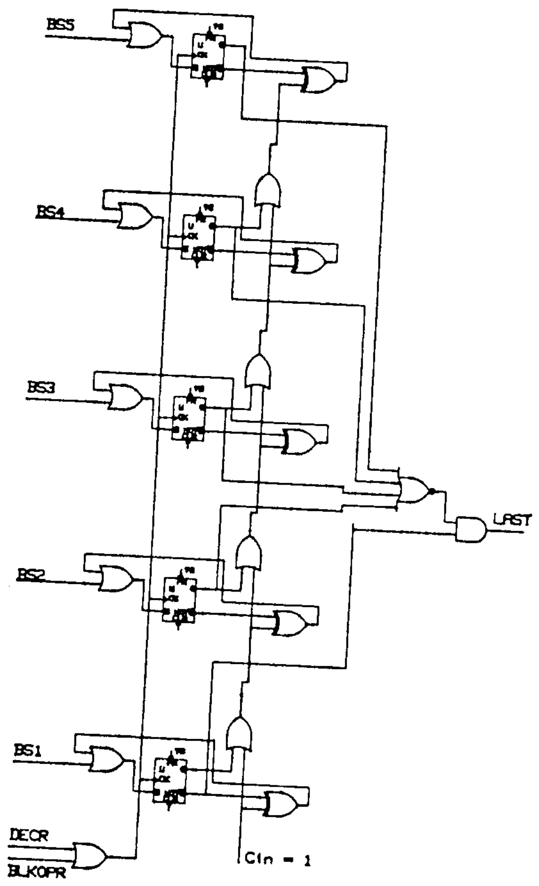


Figure 19. Block size decrementer

Once the slave NBC finishes a data transfer, it gives out a STRTWAIT signal and starts counting. When the count is reached, a signal OVER tells the NBC that it can start the next data transfer. It also resets the counter. The circuit generating a predefined waiting period is shown in Figure 20.

## 5.8 Retry logic

Whenever an operation on the NuFTbus fails, the master BIU retries it before informing the master processor about it. The operation might have to be retried due to:

- Double error in the control (acknowledgement) signals.
- Double error in the data received
- 'Try again later' acknowledgement
- 'Error' acknowledgement (This may be due to double error in the address or control signals sent by the master initially or due to some other error at the slave BIU)

The number of retries is programmable and it is loaded in the BIU command register during initialization. When the master NBC retries the operation, it increments a counter. When the count reaches the limit, the master NBC gets a RTRYFIN signal from the counter. Then it sets a flag in the BIU status register and gives out an error to the master processor. The circuit keeping count of retries is shown in Figure 21.

## 5.9 Operation type detect

Control lines $TM_1$, $AD_1$ and $AD_0$ indicate the type of operation on the NuFTbus. Slave NBC latches these control lines and uses a 3 to 8 decoder to determine the type of op-

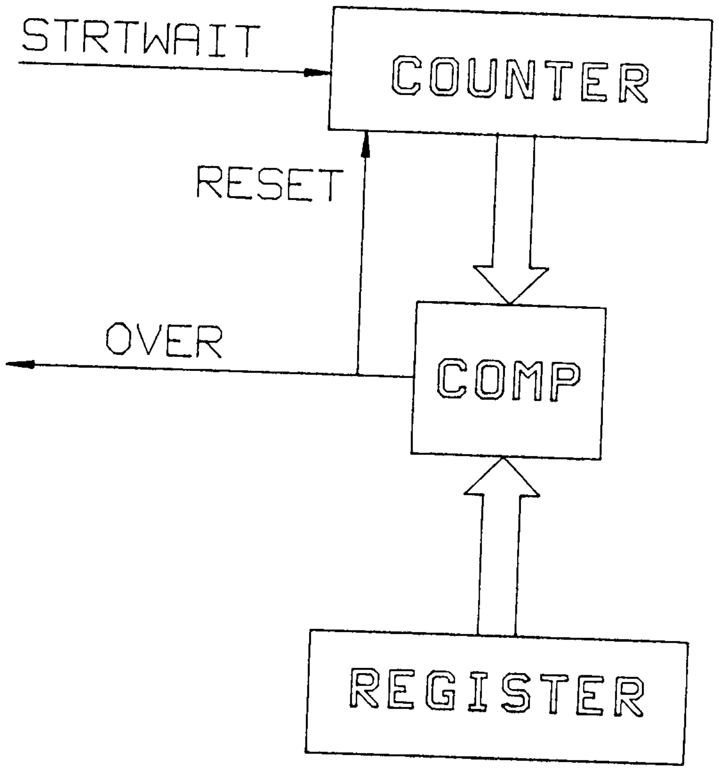


Figure 20. Circuit to wait between intermediate word transfers in block mode

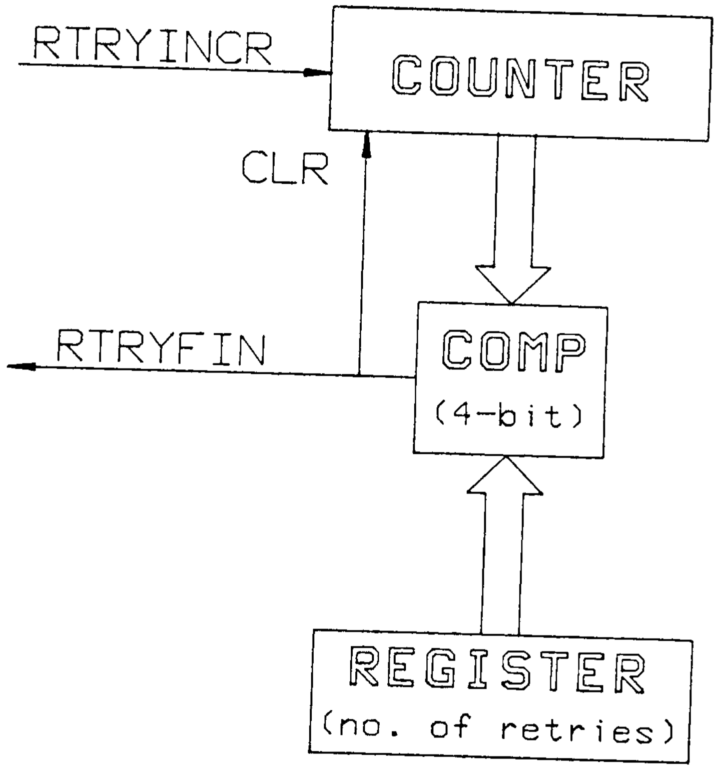


Figure 21. Retry counting circuit

eration. The master processor sends the control information to the BIU through the signals Byte/other*, $AD_1$ and $AD_0$. These are latched and decoded by the master BTC, using a 3 to 8 decoder. The slave NBC generates the acknowledgement using internal $GTM_1$ and $GTM_0$ signals, which are latched using signal LA. When the operation is detected as a byte or half word write, the incoming byte/half word is merged with the word present at that address using separate latch signals to DIREG (data in register). This is shown in Figure 22.

## 5.10 BIU register operations

Both the BTC and the NBC can access the BIU registers. A read/write operation on the BIU register initiated by the master processor is completed by the BTC, while a NuFT-bus operation doing read/write to a BIU register is completed by the NBC. The individual registers in the BIU are addressed by local decoding after address decode unit gives a high on the output BIUSPACE. The read and write operations are done using signals BIU_R_OUT and WR_BIU_REG (Figure 23).

## 5.11 Error handling

The different types of error conditions and the actions taken by the BIU corresponding to these types of errors are as follows:

1. Double error in addr/data from board: The BTC sets a flag in the BIU status register and informs the processor by raising the ERROR output.
2. Double error in data in a BIU register: The BTC sets a flag in the BIU status register and informs the processor by raising the the ERROR output.

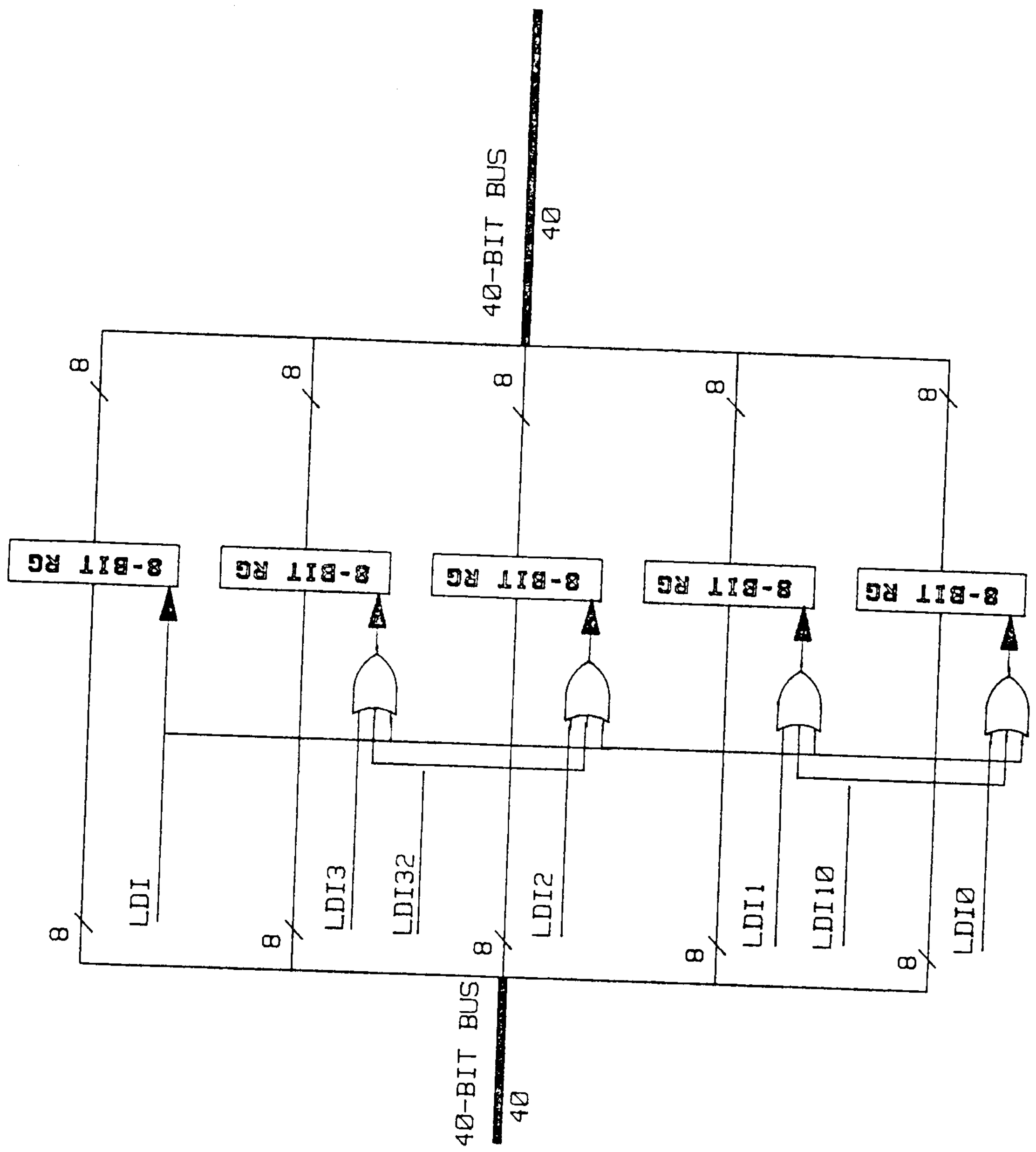


Figure 22. Read Modify Write operation using Data In Register

3. Double error in addr/data from the NuFTbus: The NBC increments the retry counter and retries the operation.
4. Double error in control from the NuFTbus: The NBC increments the retry counter and retries the operation.
5. 'Error' acknowledgement from the slave BIU: The NBC increments the retry counter and retries the operation.
6. 'Timeout' acknowledgement: The NBC sets a flag in the BIU status register and informs the processor by raising the ERROR output.
7. Lock detect: The NBC gives out an attention cycle for rearbitration.
8. ARBERR: The NBC sets a flag in the BIU status register and informs the processor by raising the ERROR output.
9. Retries fail: The NBC sets a flag in the BIU status register and informs the processor by raising the ERROR output.

The BIU status register consists of the following flags:

| | |
|---|---|
| **TMOUT** | A 'time out' acknowledgement from the slave |
| **RTRYFAIL** | All retries failed |
| **BIUMERR** | Double error in the BIU register |
| **ADDRERR** | Double error in the address from the board |
| **MEMERR** | Double error in the data from the board |
| **ARBFAIL** | Error in arbitration |

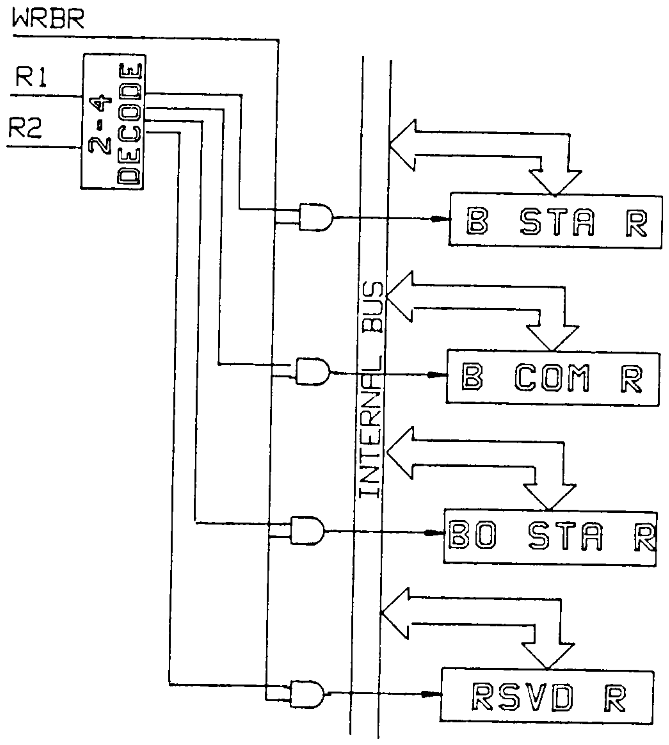


Figure 23. Read/write to BIU registers

This completes the discussion of the design of the BIU. The next chapter discusses the simulations carried out to verify the design and the simulation results.

# 6.0 SIMULATION AND SIMULATION RESULTS

## 6.1 Introduction

This chapter discusses the simulations carried out to verify the design of the BIU and gives the simulation results obtained.

The control units BTC and NBC are described functionally using VHDL. Another important part of the BIU, arbitration logic, is simulated at the gate level using the PC-LOGS simulator from Personal CAD systems, Inc. Different simulators such as PCLOGS and the VHDL simulator are used. The simulations of the arbitration unit are done on the PCLOGS due to the availability of a schematic capture front end. It is very easy to enter the gate level circuit using a schematic capture system. VHDL is a very good tool for behavior modeling. Thus, VHDL is used for modeling the bus protocols.

The outputs from the control units BTC and NBC control all the other components of the BIU. For example, to arbitrate for the control of the NuFTbus, the NBC gives a signal to the arbitration unit to start the arbitration and after 2 bus cycles (arbitration cycle period), the NBC checks the grant input from the arbitration unit. Also, for storing the data in a register, the BTC or the NBC give a latching pulse to the register and enable the outputs of the register when the data is to be put on the bus. Thus, the signals generated by the control units and responses from the arbitration unit are the interfaces between the two simulations.

## 6.2 Introduction to VHDL

VHDL is the VHSIC Hardware Description Language. Any circuit to be described in VHDL is divided into two parts, the interface description and the architectural body. The interface description has the name of the circuit and lists the input, output and in/out signals. The signal type can be single bit or bit_vector. The architectural body describes the behavior of the circuit. In the architectural body, the components can be instantiated (similar to calling a subcircuit in a hierarchy) [21].

Two major modeling elements in VHDL are the block statement and the process statement. The block has a declaration section and an executable section. The blocks can be nested. The blocks can have guard conditions associated with them. The statements inside the block are enabled only when the guard condition becomes true. For example,

```
   B1: block (X = 1)
  begin
   ---
   ---
  end block;
```

This block gets enabled only when X = 1.

```
   B1: block (ENA = '1' and not ENA'stable)
  begin
   ---
  ---
  end block;
```

ENA = '1' and not ENA'stable implies that, ENA is 1 and ENA has just changed its state (indicated by not ENA'stable), i.e. a rising edge on the ENA signal. Thus, this block gets enabled on a rising edge on the ENA line.

The process construct is used to represent concurrent activities. Each process has a sensitivity list. When a signal in the sensitivity list changes (from 1 to 0 or vice versa), the process is enabled and the statements in the process are executed.

VHDL supports different data types such as boolean, bit, bit_vector, integer and character. Functions and procedures can be defined in VHDL. VHDL supports numerous control constructs similar to those in many high level languages. Some of them are IF-THEN-ELSE, CASE, LOOP, RETURN, NEXT, WAIT and EXIT. Thus the VHDL is a versatile hardware description language with many features of a high level language.

### 6.2.1 Implementing BTC and NBC using VHDL

The BTC and the NBC are described as separate entities in VHDL description. The BTC interfaces with the board, the NBC and other components of the BIU, while the NBC interfaces with the NuFTbus, the BTC and other components of the BIU. A 40 MHz system clock is defined for the internal operations of the BIU and a 10 MHz clock with unsymmetric duty cycle is defined for the NuFTbus operations.

The VHDL code for the NBC as well as for the BTC is written as one big process statement. The process is enabled on the rising edge of the system clock. The different states of the state machine are represented by IF statements. The state transitions are implemented using dummy signals. For example, when the program is in a particular state, after executing the statements in that state, it sets a dummy signal, DUMMY1. The if statement for the next state is

```
    if DUMMY1 = '1' and not DUMMY1'stable(25 ns) then
 ---
 ---
    end if;
```

This implies that the if condition is true when DUMMY1 is one and DUMMY1 is not stable for the last 25 ns (25 ns is the period of the system clock), i.e., DUMMY1 has changed to 1 within the last 25 ns. Consider the following example.

```
    process (clock)
        if Z = '1' and not Z'stable(25 ns) then
     ---
          X <= '1';
          DUMMY1 <= '1';
   end if;
        if DUMMY1 = '1' and not DUMMY1'stable(25 ns) then
     ---
     ---
          DUMMY2 <= '1';
   end if;
        if DUMMY2 = '1' and not DUMMY2'stable(25 ns) then
     ---
     ---
          DUMMY3 <= '1';
   end if;
```

```
        if DUMMY3 = '1' and not DUMMY3'stable(25 ns) then
      ---
      ---
          DUMMY4 <= '1';
    end if;
  end process;
```

After the Z goes to one, with the next rising edge of the clock, X goes to one and DUMMY1 also goes to one. With the next clock pulse, the next if statement

```
  if (DUMMY1 = '1' and not DUMMY1'stable(25ns) ) then
```

becomes true, which sets DUMMY2. At the third clock pulse, the next if statement gets executed and so on. Thus the different states are linked together by the dummy signals. Both state machines are implemented in this manner. In the case of the NBC, rising and falling edges of the bus clock are also used for certain operations, along with the edges of the internal clock.

Both the NBC and the BTC are tested individually for different operations. Then the BTC and the NBC are connected together to form a BIU and its functionality is tested by inputting signals from the processor (board) side and the NuFTbus side. Then two BIUs are connected together to simulate different transactions on the bus. Input signals from the master processor and responses from the slave processor are given in the test_bench file and the transactions on the NuFTbus are simulated. The simulation results for single word read, single word write, block read, block write and concurrent master-slave operation are given in the next section.

### 6.2.2 Results from the VHDL simulation

This section explains the results from the VHDL simulations carried out for different BIU operations. The operations are single word and block read, single word and block write and overlapping master/slave operations. There are three Figures (plots of timing diagrams) for each operation. These plots show the signals on the board-BIU interface on the master and on the slave sides, signals on the NBC-BTC interface in master and slave BIUs and signals from the NuFTbus (START, ACK and $TM_0$). Every transaction is started from the master processor. Input signals from the master processor are given through the test_bench file in the VHDL simulation (e.g., SENI, BLKEN). The responses from the slave processor side are also given in the test_bench file (e.g., RELACK, WAITI). All the internal BIU signals and the signals on the NuFTbus are generated in the simulation.

The block operation results for two and three word transfers are shown here. The rest of the block transfer transactions are similar.

#### *6.2.2.1 Single Word Read*

SENI (single enable in) and BLKEN (block enable) are the inputs from the processor. Initially both are high. SENI going low and a high on BLKEN starts a single data transfer operation. (Figure 24). A high on RWBO indicates that it is a read operation (RWBO shown in the Figure). The BTC latches the address and decodes the operation and determines that it should start a corresponding NuFTbus transaction. So the BTC puts the processor in a wait state by raising WAITO and raises BNLOCK to tell the NBC that a master mode operation is to be carried out. The NBC sends an acknowledgement to BTC with raising BNACK signal. Then BTC sets TOR (transfer out ready) to inform the NBC that it should continue the operation. The NBC responds with a high on TIP (transfer in progress). The BTC lowers TOR and waits for a high on DIR (data in ready) input from the NBC. Once the NBC finishes fetching the data word from the slave, it raises DIR. Then, the BTC puts the received data word on the board-BIU interface and lowers WAITO to the processor. SENI going high tells the BTC that the data has been accepted by the processor. So the BTC lowers BNLOCK to the NBC. In response, the NBC lowers its acknowledgement BNACK and then both controllers go into their initial states awaiting the next operation.

The second Figure (Figure 25) for single word read operation shows the signals on the slave side. The slave NBC asks for the release of the local processor bus by raising REL. Once it gets a RELACK, it sets BBLOCK to tell the slave BTC that a slave mode operation is to be carried out. With a high on BBACK, the NBC sets TIP and AIR (address in ready) signals. Then the BTC places the received address on the board-BIU interface and lowers SENO. RWBI (read write board in) is high which is not shown in Figure. The processor places the data on the data bus. The data is latched by the BTC (If the decode and read operations on the board are slow then WAITI goes high to tell the BTC to wait. Then the BTC latches the data word after WAITI line goes low). In this example WAITI is low all the time. Once the BTC receives the data word, it lowers TOR. The NBC acknowledges this by lowering AIR and sends the data on the NuFTbus. Then it lowers REL, BBLOCK and TIP signals and goes to initial state. The BTC lowers BBACK and also goes to its initial state.

The third figure for single word read operation (Figure 26) shows some of the signals on the NuFTbus. START and ACK lines are bidirectional. In this simulation, STARTI represents START in the input mode while STARTO represents START in the output mode. The same is true for ACK. Thus, STARTO and ACKO are the lines going out from the master BIU, whereas STARTI and ACKI are the lines coming into the master BIU.

A low on STARTO (and a high on ACKO) for one bus clock indicates a start cycle generated by the NBC on the master side, and a low on ACKI (and a high on STARTI) indicates the end of the NuFTbus transaction, which is generated by the NBC on the slave side.

### *6.2.2.2 Single Word Write*

SENI going low and a high on BLKEN starts a single data transfer operation. (Figure 27). A low on RWBO indicates that it is a write operation. (RWBO is not shown in the Figure). The BTC latches the address and decodes the operation, and determines that it should start a corresponding NuFTbus transaction. So, the BTC puts the processor in a wait state by raising WAITO, and raises BNLOCK to inform the NBC that a master mode operation is to be completed. The NBC acknowledges it with raising BNACK signal. Then the BTC sets TOR (transfer out ready) to tell the NBC to continue the operation. The NBC responds with a high on TIP (transfer in progress) and AIR (address in ready). The BTC lowers TOR and waits for AIR to go low. This tells the BTC that data has been written properly. Once the NBC finishes the NuFTbus transaction, it lowers AIR. Then the BTC lowers the WAITO to the processor. SENI going high indicates the end of the operation. So the BTC lowers BNLOCK to the NBC. In response the NBC lowers its acknowledgement, BNACK, and also lowers TIP. DIR

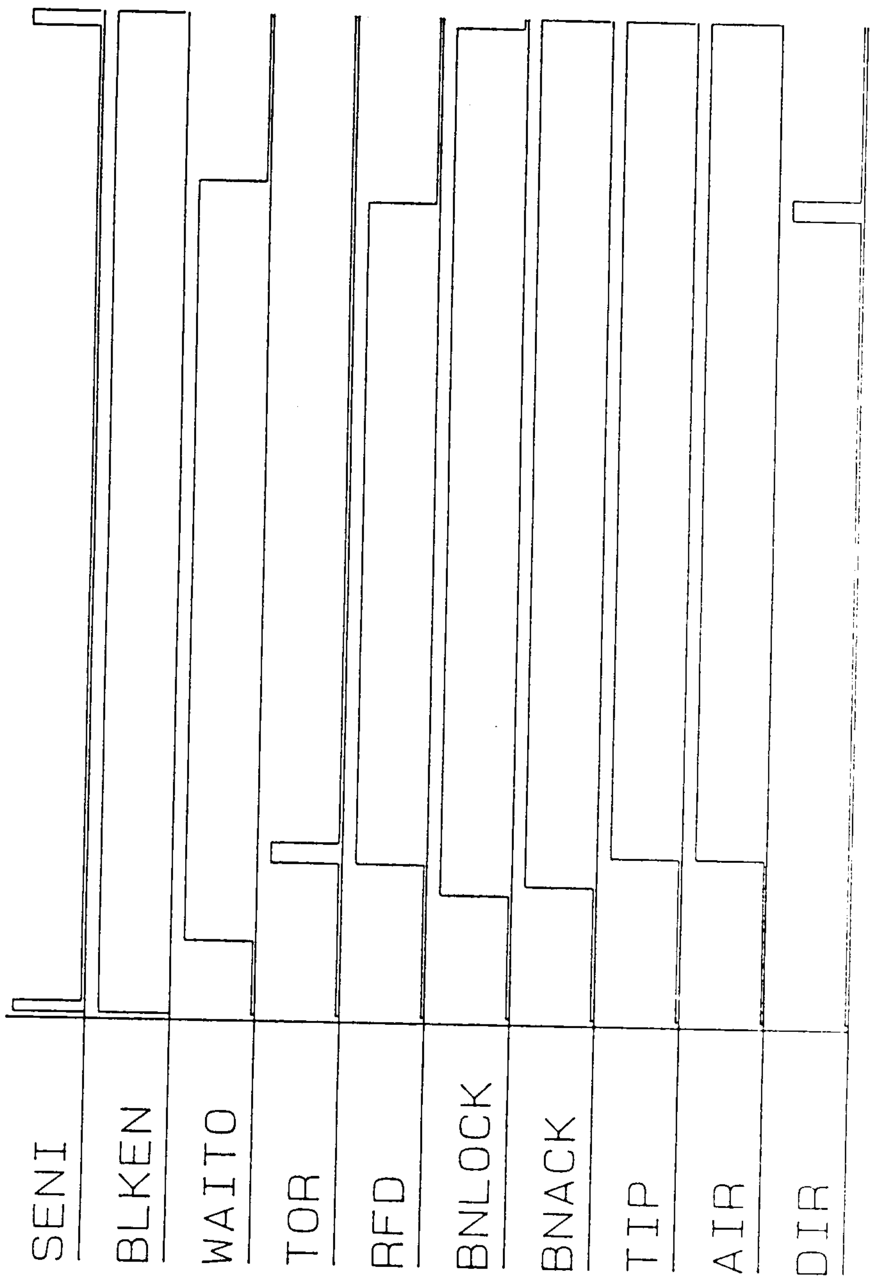


**Figure 24. Single read operation**

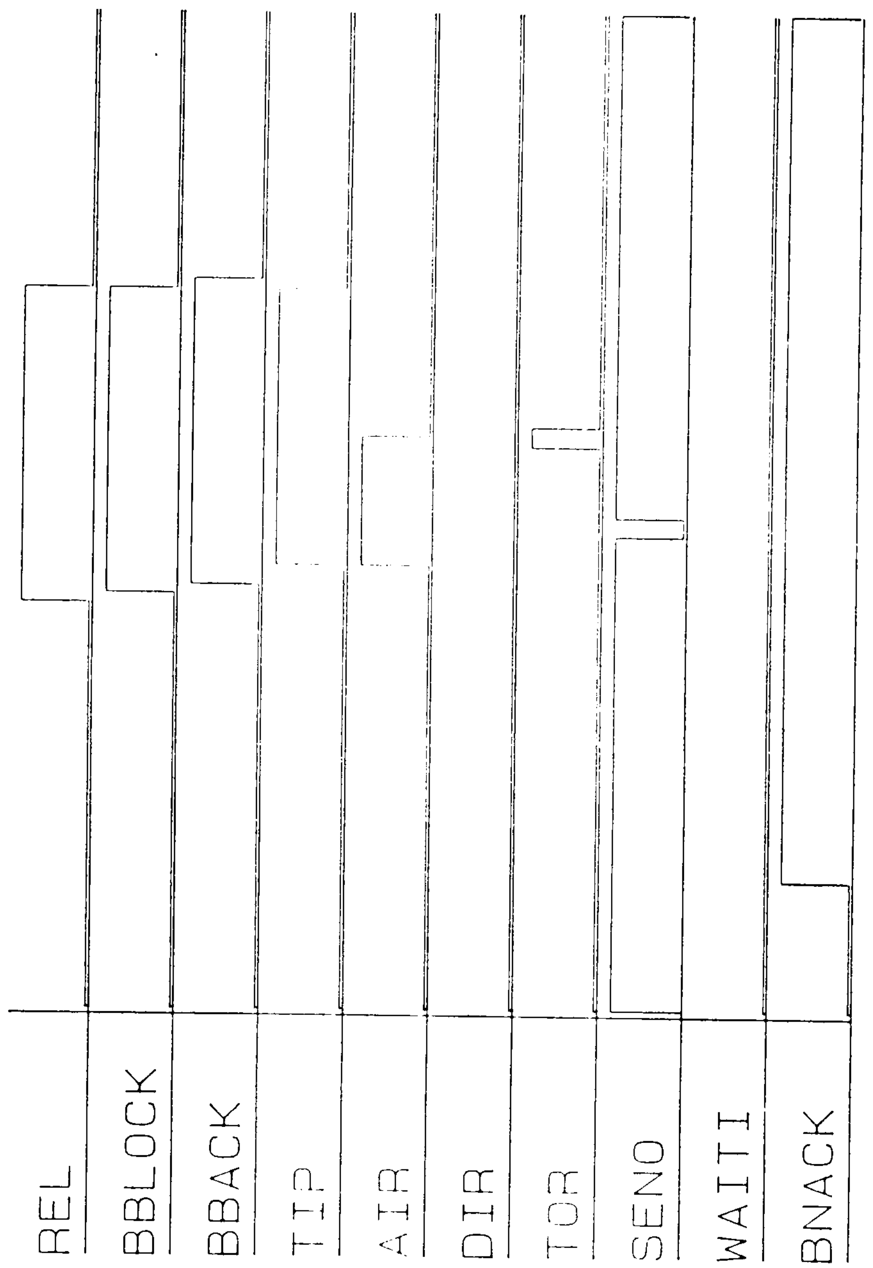


Figure 25. Single read operation (continued)

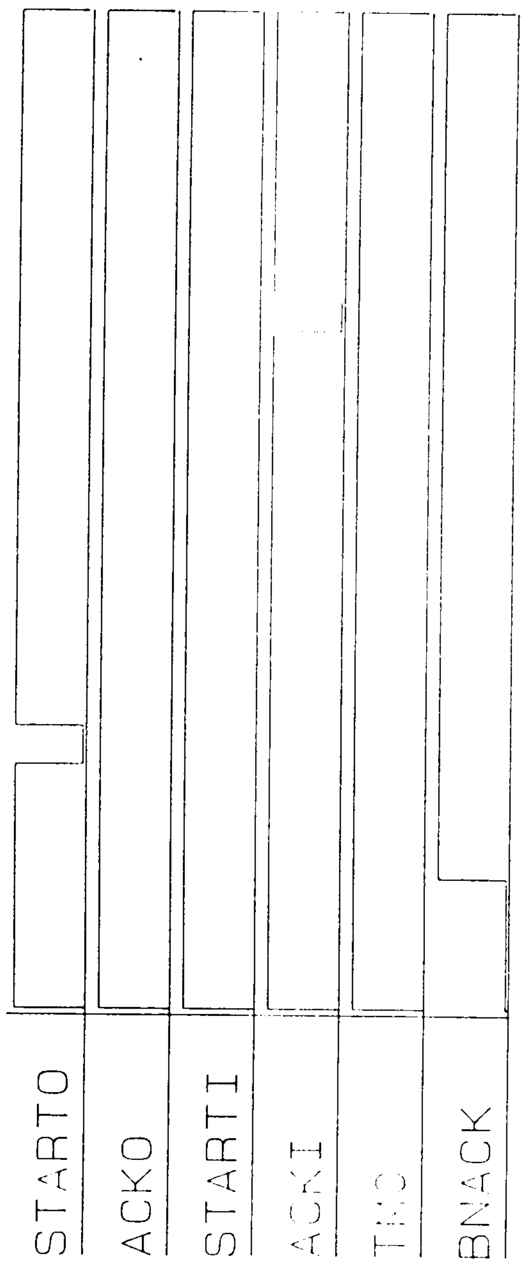


Figure 26. Single read operation (continued)

signal in not used in this operation. Then both controllers go to their initial states awaiting the next operation.

The second figure (Figure 28) for single word write operation shows the signals on the slave side. The slave NBC asks for the release of the local processor bus by raising REL. Once it gets a RELACK, it sets BBLOCK to inform the slave BTC that a slave mode operation is to be carried out. With a high on BBACK, the NBC sets TIP, AIR (address in ready) and DIR (data in ready) signals. A read-modify-write operation is shown in this example instead of a normal write operation. If a byte or a half word is to be written, then the BTC reads the word at that address, changes the particular byte or half word and writes the new word back. So, the BTC places the received address on the board-BIU interface and lowers SENO (RWBO is high). The processor places the data on the data bus, which is latched by the BTC. Once the BTC has the data word, it modifies the byte or half word, places the modified word on the board-BIU interface and lowers SENO again (This time RWBO is low, for write operation). (If the board logic is slow, the input WAITI in raised. Then the BTC holds the data on the interface till WAITO goes low). The BTC raises TOR to indicate the end of write operation. Then the NBC lowers AIR and DIR. The BTC lowers TOR in response. Once the NBC sends out an acknowledgement on the NuFTbus, it lowers REL, BBLOCK and TIP. Then the BTC lowers BBACK and both controllers go to their respective initial states.

The third figure for single word write operation (Figure 29) shows some of the signals on the NuFTbus. A low on STARTO (ACKO is high) for one bus clock indicates a start cycle generated by the NBC on the master side, and a low on ACKI (and a high on STARTI) indicates the end of the NuFTbus transaction which is generated by the NBC on the slave side.

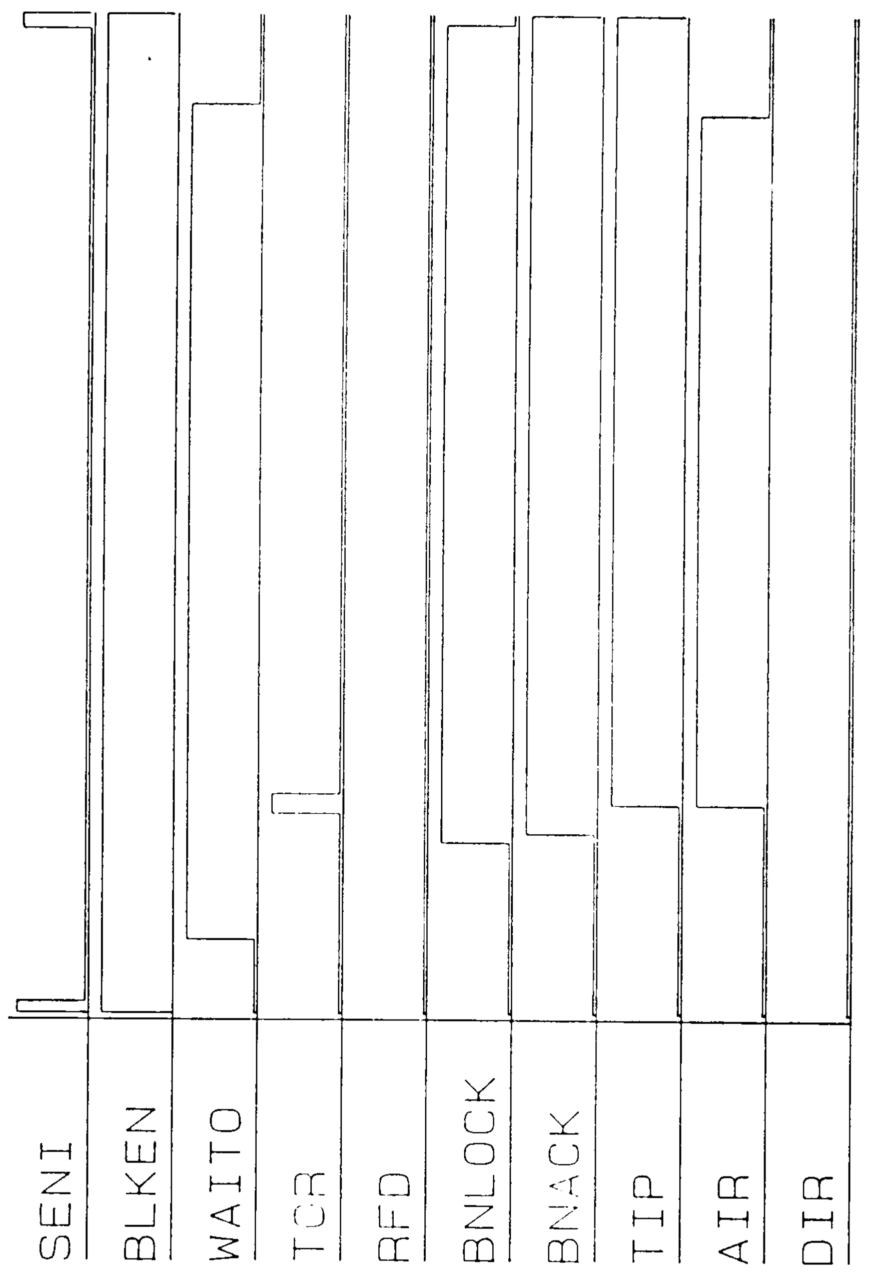


Figure 27. Single write operation

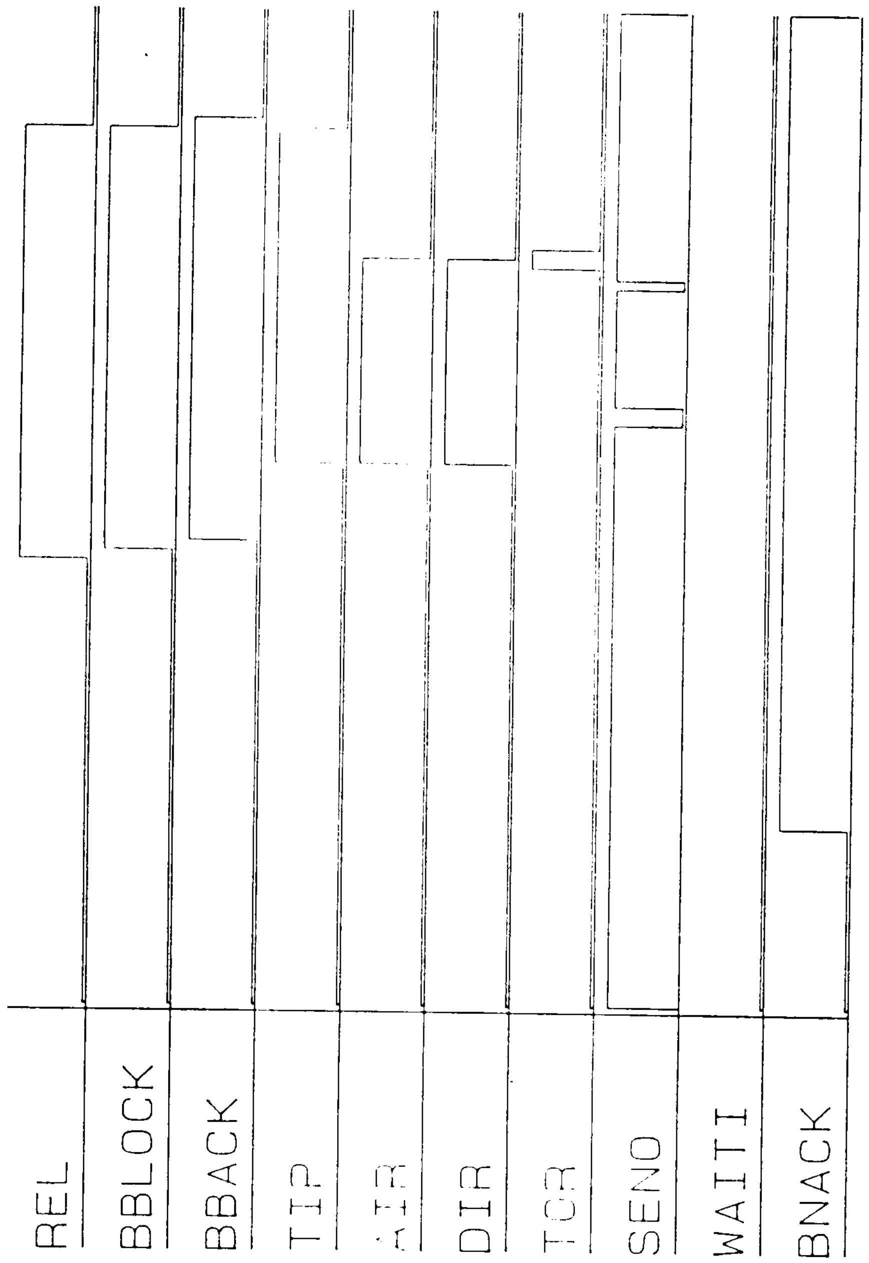


Figure 28. Single write operation (continued)

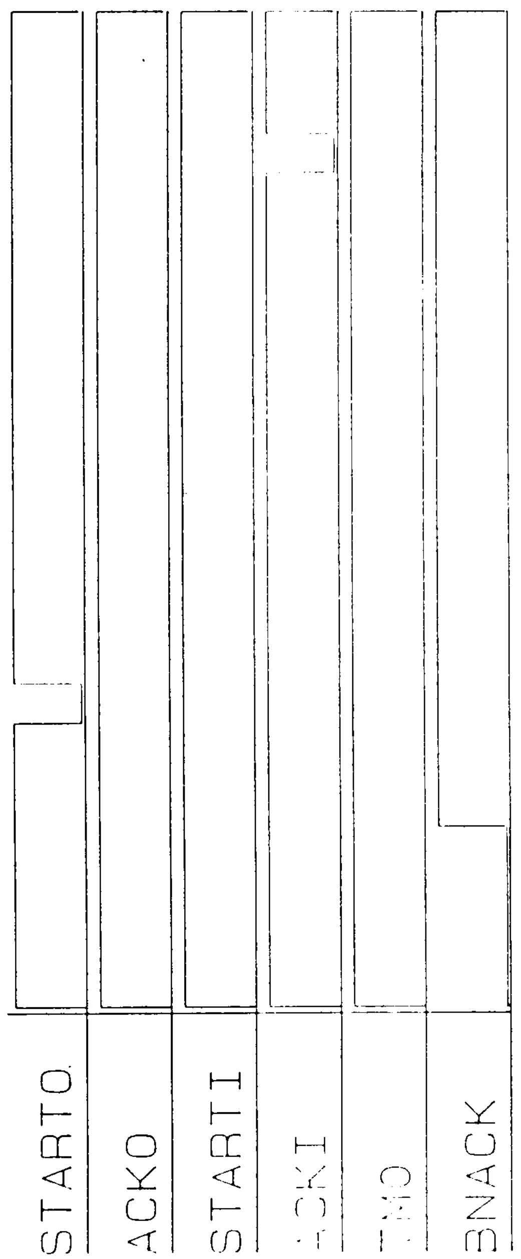


Figure 29. Single write operation (continued)

### *6.2.2.3 Overlapping master/slave operation*

As explained in the previous chapter, the NuFTbus transactions (i.e., slave mode transactions) have priority over the transaction started by the master processor (i.e., master mode transaction). In Figure 30, SENI goes low (BLKEN is high) to indicate the start of a master mode transaction (RWBO is high to indicate a read operation). The BTC puts the processor in the wait state by raising WAITO and sets BNLOCK to inform the NBC to start a master mode operation (Figure 31). But instead of giving a high on BNACK, the NBC sets BBLOCK signal. This informs the BTC that the NBC has a slave mode operation. As the slave mode operation has a priority over the master mode operation, the BTC suspends the master read operation. It gives out a high on BBACK to tell the NBC that it is ready to complete the slave mode operation first. Then the NBC raises TIP and AIR. The BTC puts the address on the board-BIU interface and lowers SENO (Figure 30). RWBI is high. The board is not ready, so it makes WAITI high till it puts the data word on the interface. Then the BTC makes SENO high again and raises TOR to tell that the data has been read. The NBC lowers AIR (so TOR goes to low) and gives out an acknowledgement on the NuFTbus. Then the NBC lowers BBLOCK and TIP. Then the BTC lowers BBACK. Both the NBC and the BTC check the flags and find that a master side operation is on hold (the BTC has already set BNLOCK high). So the BTC waits for BNACK from the NBC. The NBC gives out a high on BNACK. Then the BTC raises TOR and tells the NBC to proceed with the master side operation. The BTC raises TIP and AIR and starts a NuFTbus transaction. The NBC raises RFD (ready for data) and waits for DIR to go high. Once the NBC finishes the read operation and gets the data word, it raises DIR. With DIR going high, the BTC lowers RFD, places the data on the board-BIU interface, and lowers WAITO to the processor. Once SENI goes high, the BTC lowers BNLOCK. The NBC lowers BNACK, and then both controllers go to their respective initial states.

Figure 32 shows some more signals for the NBC. STARTI going low indicates the start of a slave mode transaction. The NBC asks for the release of the local processor bus by raising REL. Once the read operation is done, the NBC generates an acknowledgement by lowering ACKO for one bus clock cycle. Then the NBC starts the master mode operation by generating a START cycle (STARTO goes low). The acknowledgement comes with a low on ACKI. Thus both the operations are completed.

This explains the overlapped operation from the point of view of the BIU on one side. Figure 33, Figure 34 and Figure 35 show the signals on the side of the other BIU. As far as this BIU is concerned, the two operations are not overlapped. This BIU completes a master read operation and gets a slave read operation just after that. So, the timing diagrams look as if the master read and slave read diagrams are placed next to one another. The operations on this BIU side are as follows.

SENI goes low and RWBO is high to indicate a master read operation. The BTC raises WAITO to the processor and locks NBC by giving out BNLOCK. With a high on BNACK, the BTC raises TOR. The NBC starts the operation by setting TIP and AIR. It starts a corresponding NuFTbus transaction (Figure 35) by a low on STARTO. Once the NuFTbus transaction is over (with a low on ACKI), the NBC sets DIR. The BTC then places the data on the board-BIU interface and lowers WAITO. With SENI going high, the BTC lowers BNLOCK and goes to the initial state. The NBC lowers BNACK and goes to its initial state.

The NBC gets a low on STARTI. Then it asks for the release of the local processor bus and sets BBLOCK. With a high on BBACK, the NBC sets TIP and AIR. The BTC reads the data from the board by lowering SENO. Once the data is read, the BTC informs the NBC by raising TOR. The NBC acknowledges by lowering AIR. The NBC sends out an acknowledgement on the NuFTbus (Figure 35) by lowering ACKO. Then it lowers REL, TIP and BBLOCK and goes to the initial state. The BTC lowers BBACK and also goes to its initial state.

#### *6.2.2.4 Block Read*

SENI and BLKEN both going low (and a high on RWBO) indicate the start of a block read operation. Signal transitions on the board-BIU interface in case of a block read operation are the same as those in the case of a single read operation. The difference is in the function of SENI and BLKEN lines. After each of the intermediate words is read from the slave and passed on to the board, BLKEN goes high (instead of SENI) to indicate that the board has received the word. SENI remains low throughout the block transfer operation. BLKEN goes low again to start the next word transfer. At the end of the block transfer, both SENI and BLKEN go high at the same time to indicate the end of the block operation. These signals are shown in Figure 36. The signals on the BTC-NBC interface are similar to those in the case of single read operation. Here, TIP and AIR remain high throughout the length of the transaction. DIR going high indicates the arrival of intermediate words on the NuFTbus.

The signals for the slave BIU are shown in Figure 37. The block start address in incremented by the slave NBC and it is passed to the BTC. (The intermediate addresses are not sent from the master to the slave on the NuFTbus). Figure 37 shows the WAITI

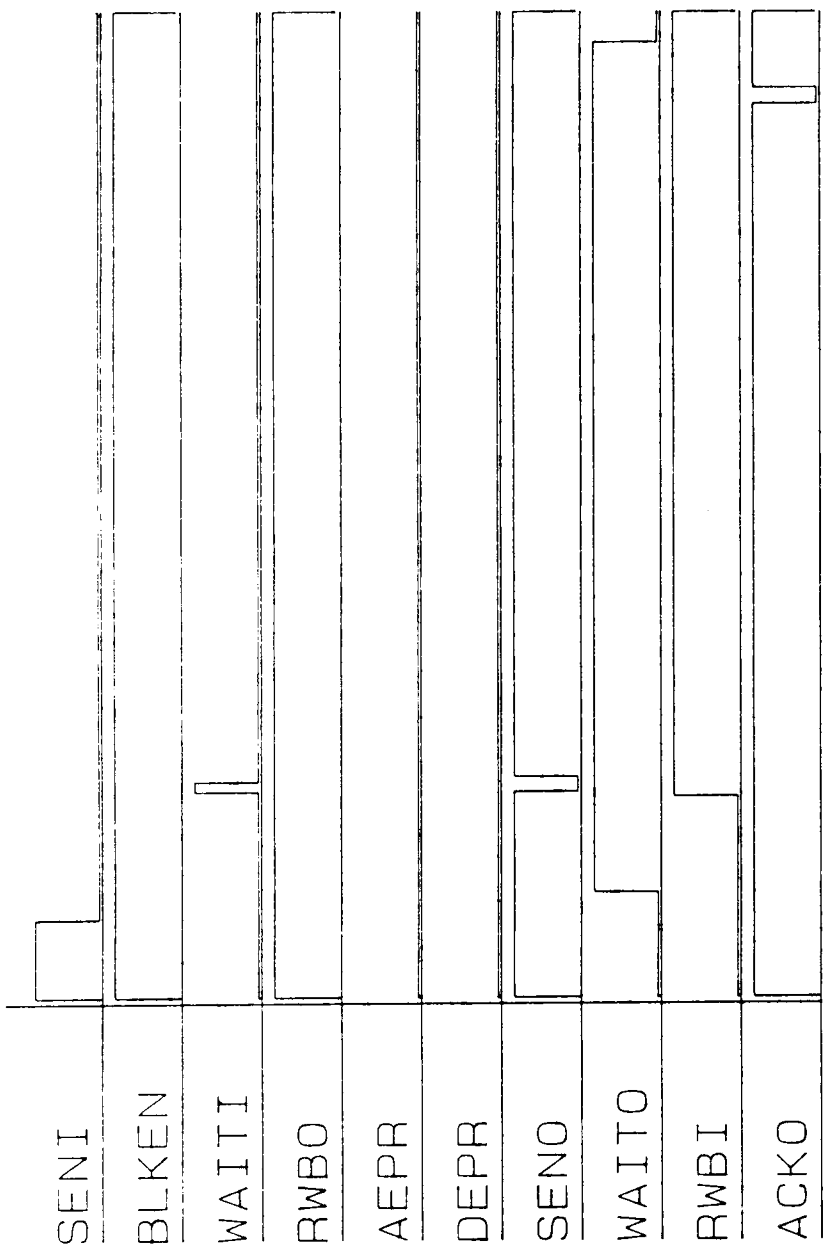


Figure 30. Overlapped master and slave side operation

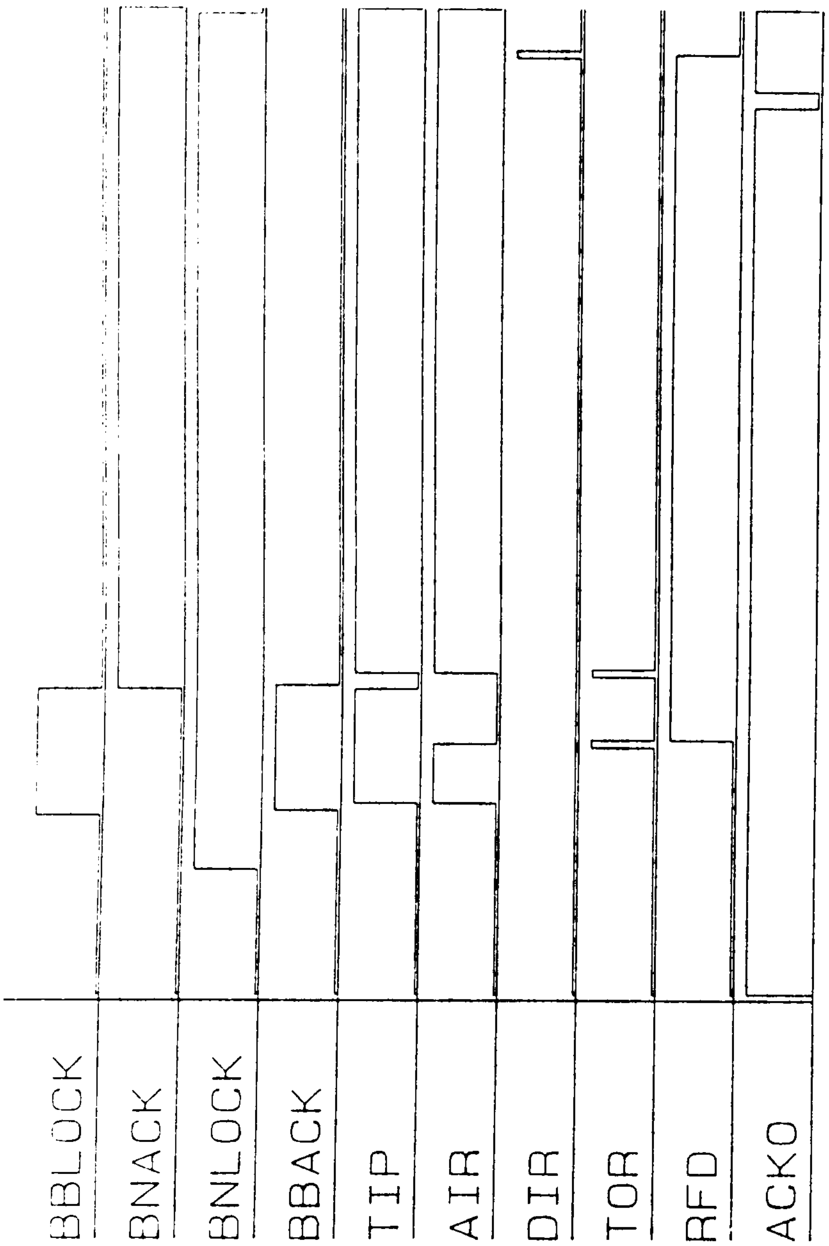


Figure 31. Overlapped master and slave side operation (continued)

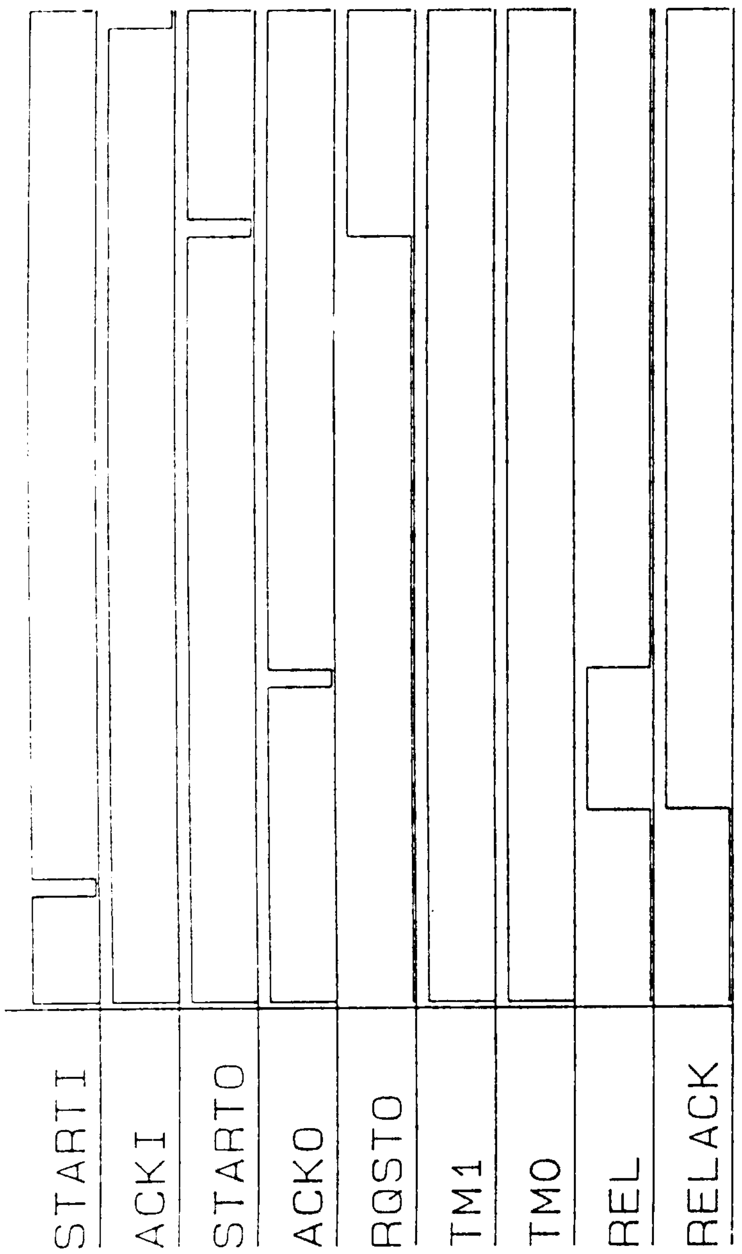


Figure 32. Overlapped master and slave side operation (continued)

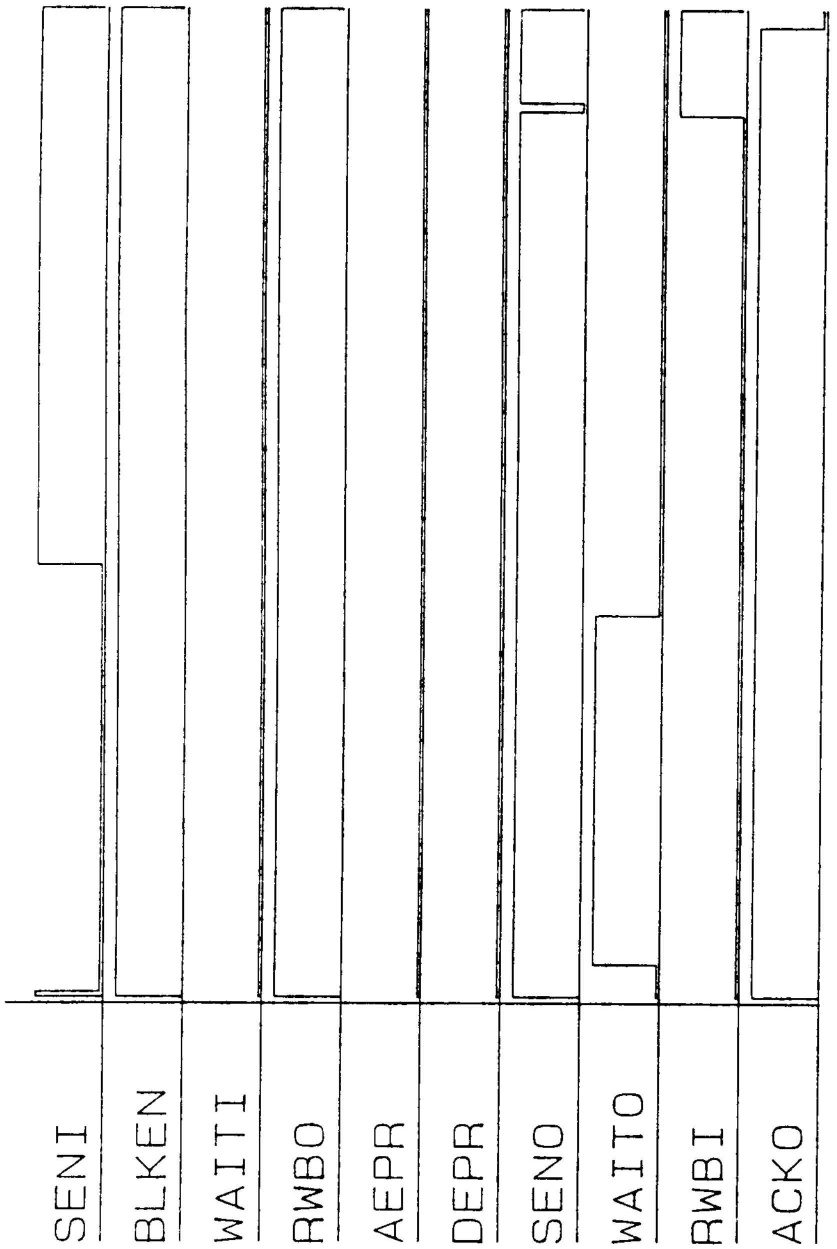


Figure 33. Overlapped master and slave side operation (continued)

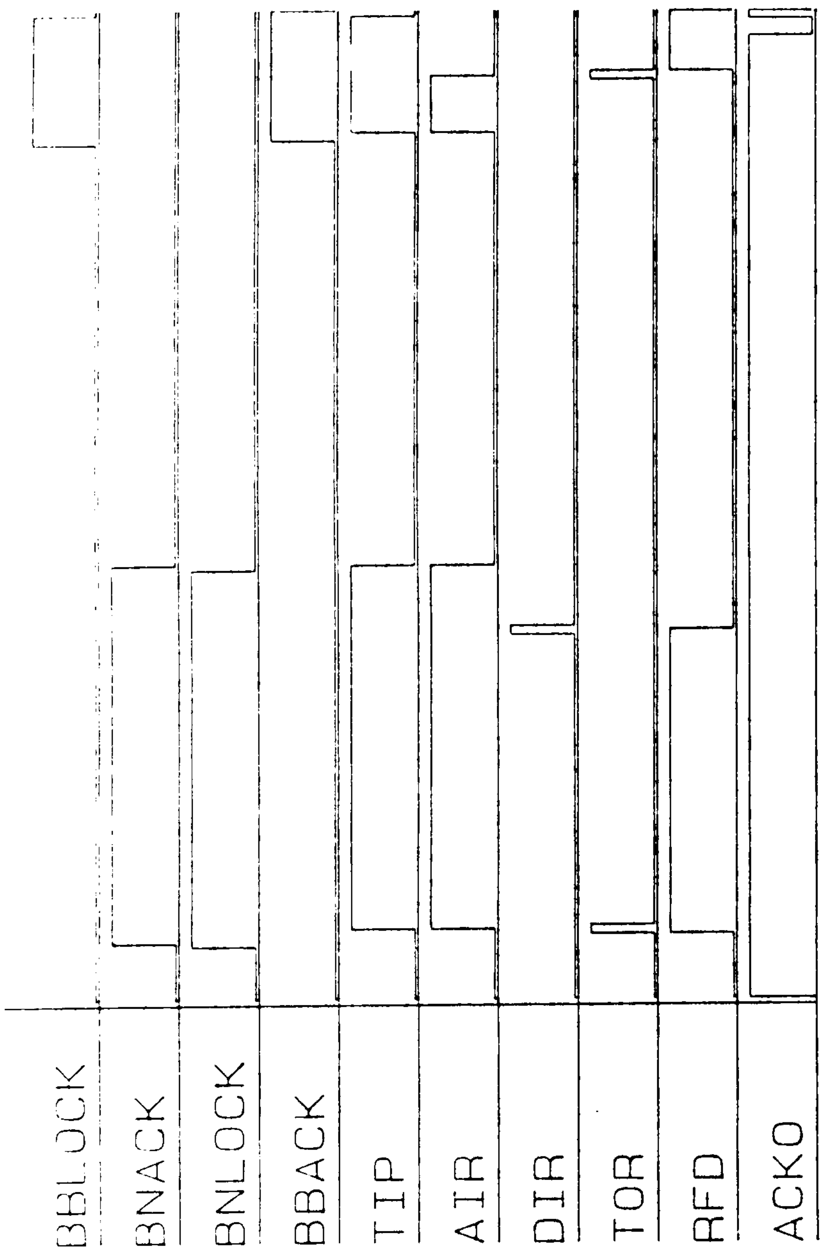


Figure 34. Overlapped master and slave side operation (continued)

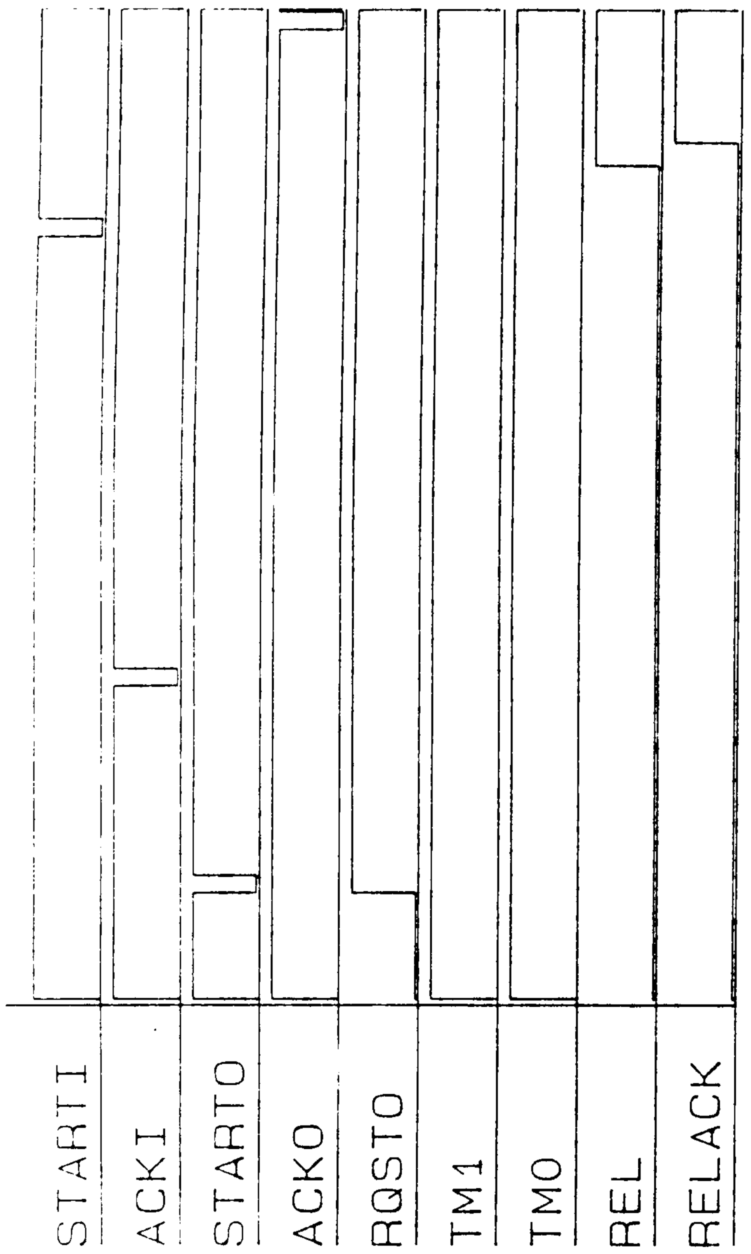


Figure 35. Overlapped master and slave side operation (continued)

signal going high till the data read operation on the board is done. SENO remains low till WAITI is high.

Figure 38 shows some of the NuFTbus signals. STARTO goes low for a period of one bus clock width indicates the start cycle. The intermediate acknowledgements are given on $TM_0$. $TM_0$ goes low to indicate to the master that the last data transfer was carried out successfully. At the end of the block transfer, ACKI goes low to indicate the end of the block read transaction.

Figure 39, Figure 40 and Figure 41 show a block read transaction for transfer of three words.

#### *6.2.2.5 Block Write*

SENI and BLKEN both going low (and a low on RWBO) indicate the start of a block write operation. The signal transitions on the board-BIU interface in case of a block write operation are the same as those in the case of a single write operation. The difference is in the function of the SENI and BLKEN lines. After each of the intermediate words is written to the slave and WAITI is lowered, BLKEN goes high (instead of SENI) to indicate the acknowledgement of the slave. SENI remains low throughout the block transfer operation. BLKEN goes low again to start the next word transfer. At the end of the block, both SENI and BLKEN go high at the same time to indicate the end of the block operation. These signals are shown in Figure 42. The signals on the BTC-NBC interface are also similar to those present in case of a single word write operation. Here, TIP remains high throughout the length of the transaction. AIR going low indicates the arrival of an intermediate acknowledgement on the NuFTbus.

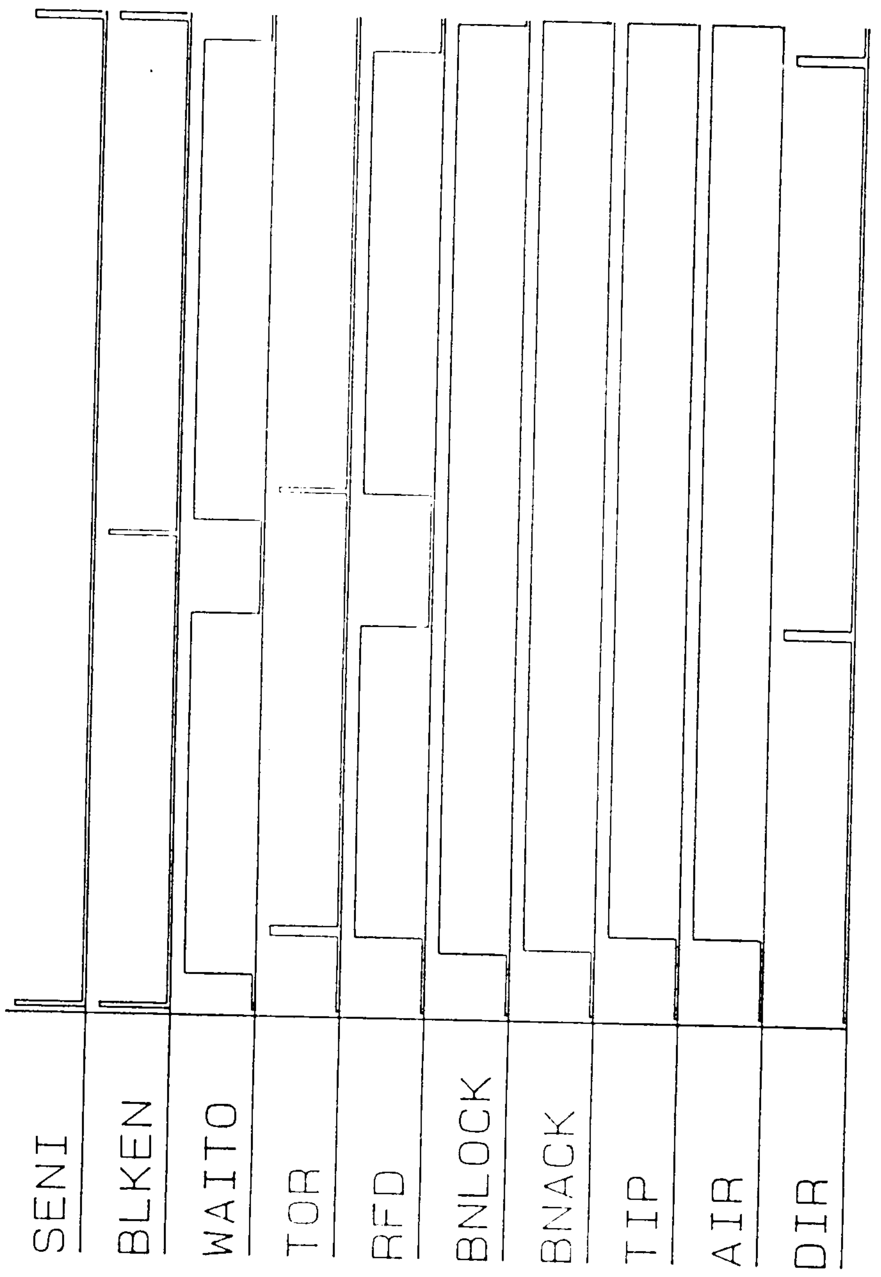


**Figure 36. Block read operation**

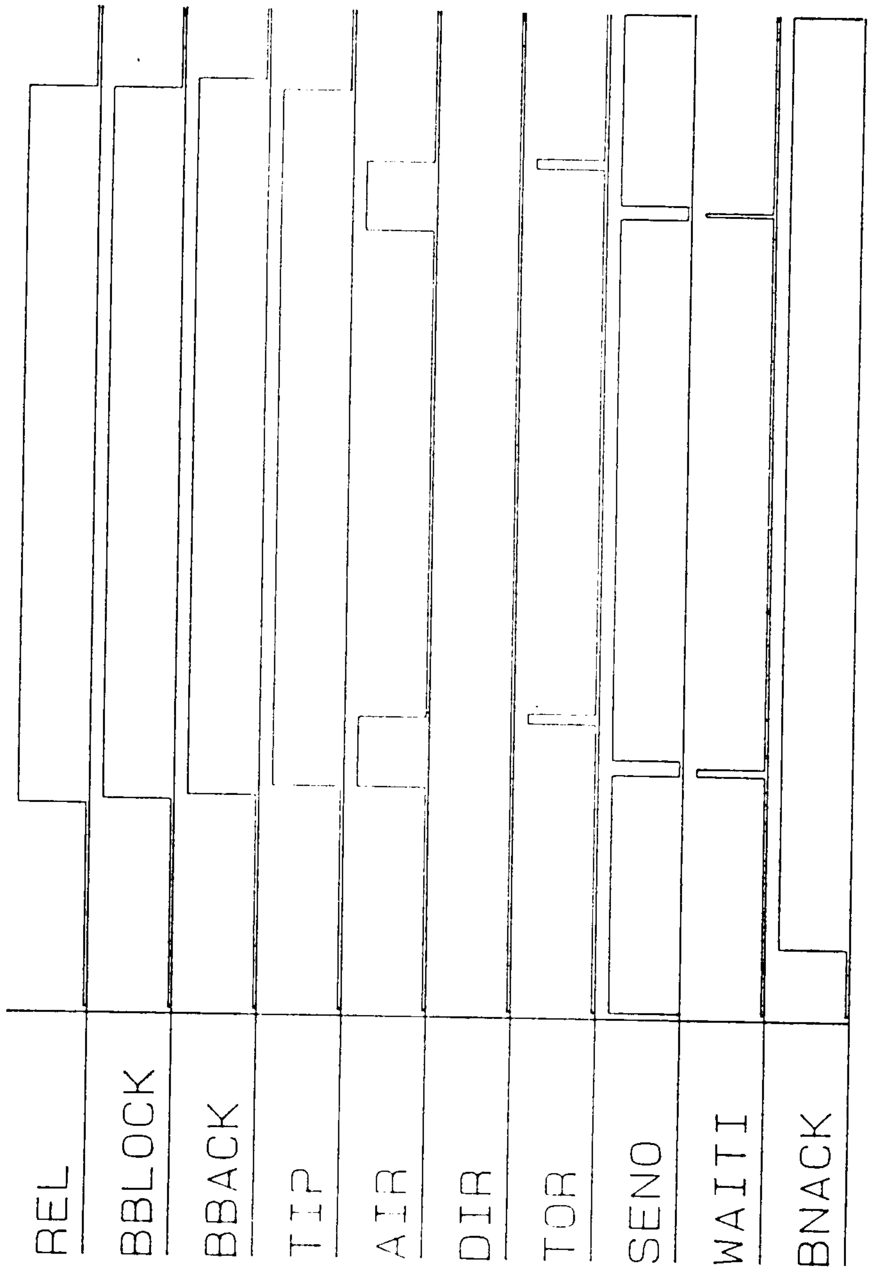


Figure 37. Block read operation (continued)

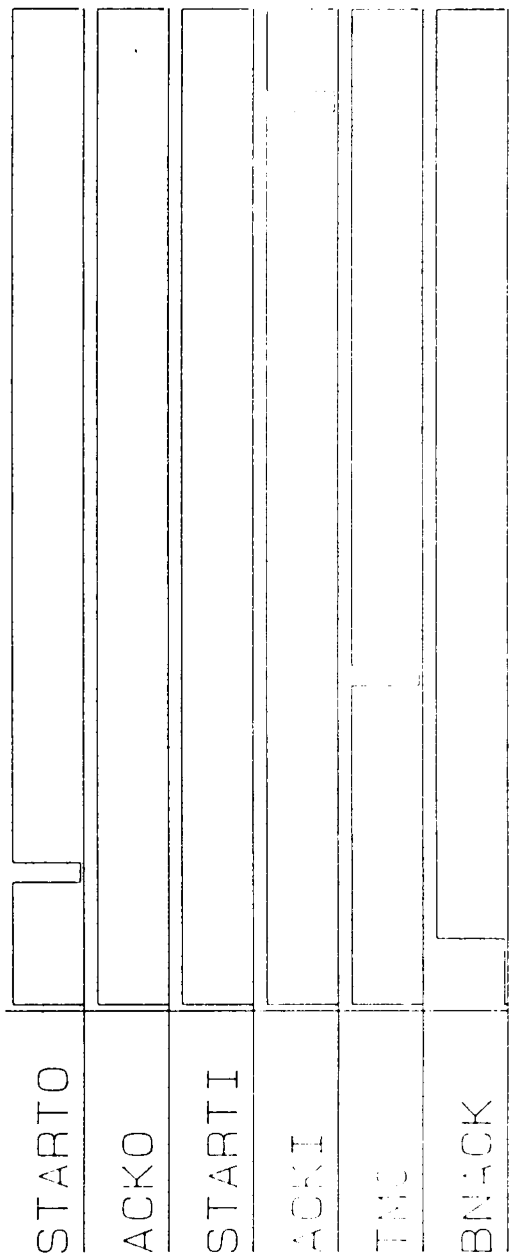


Figure 38. Block read operation (continued)

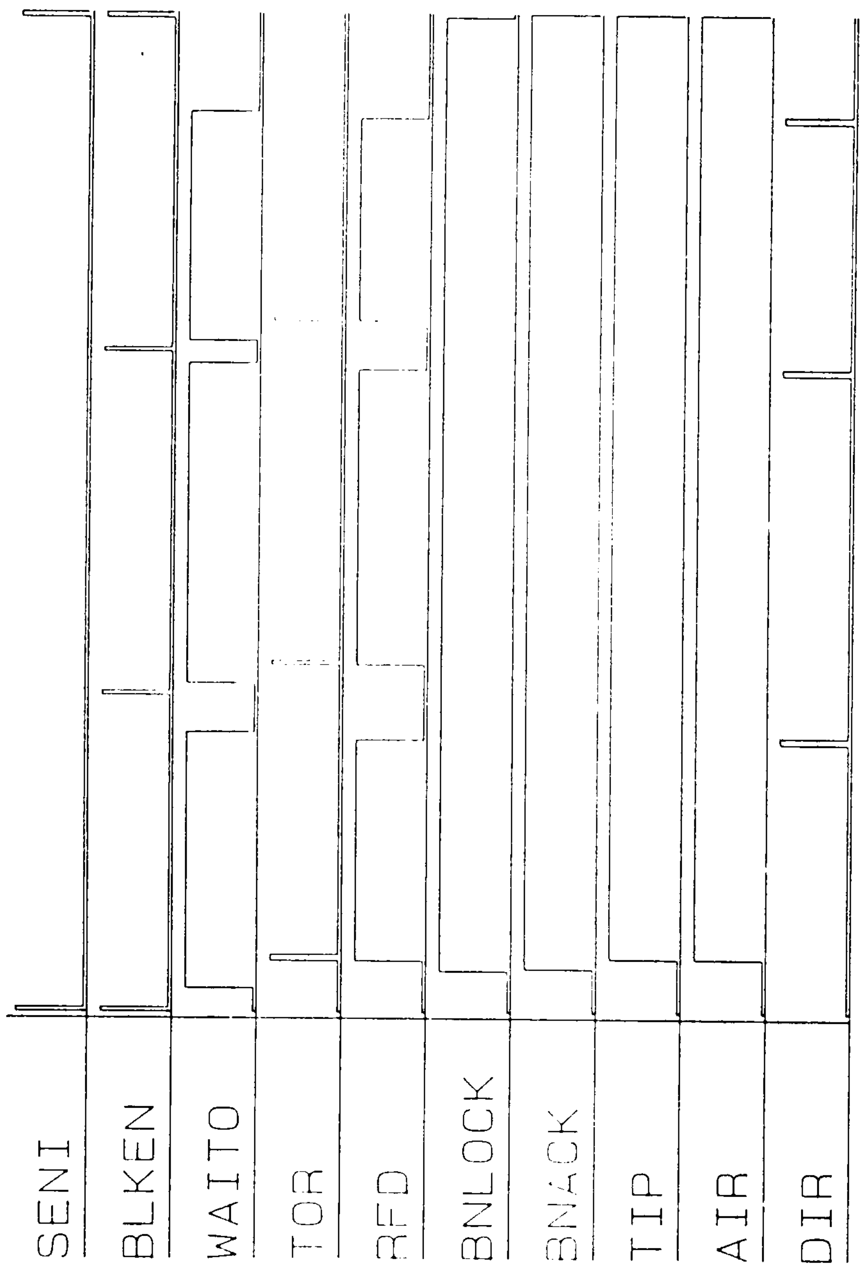


Figure 39. Block read operation for three words

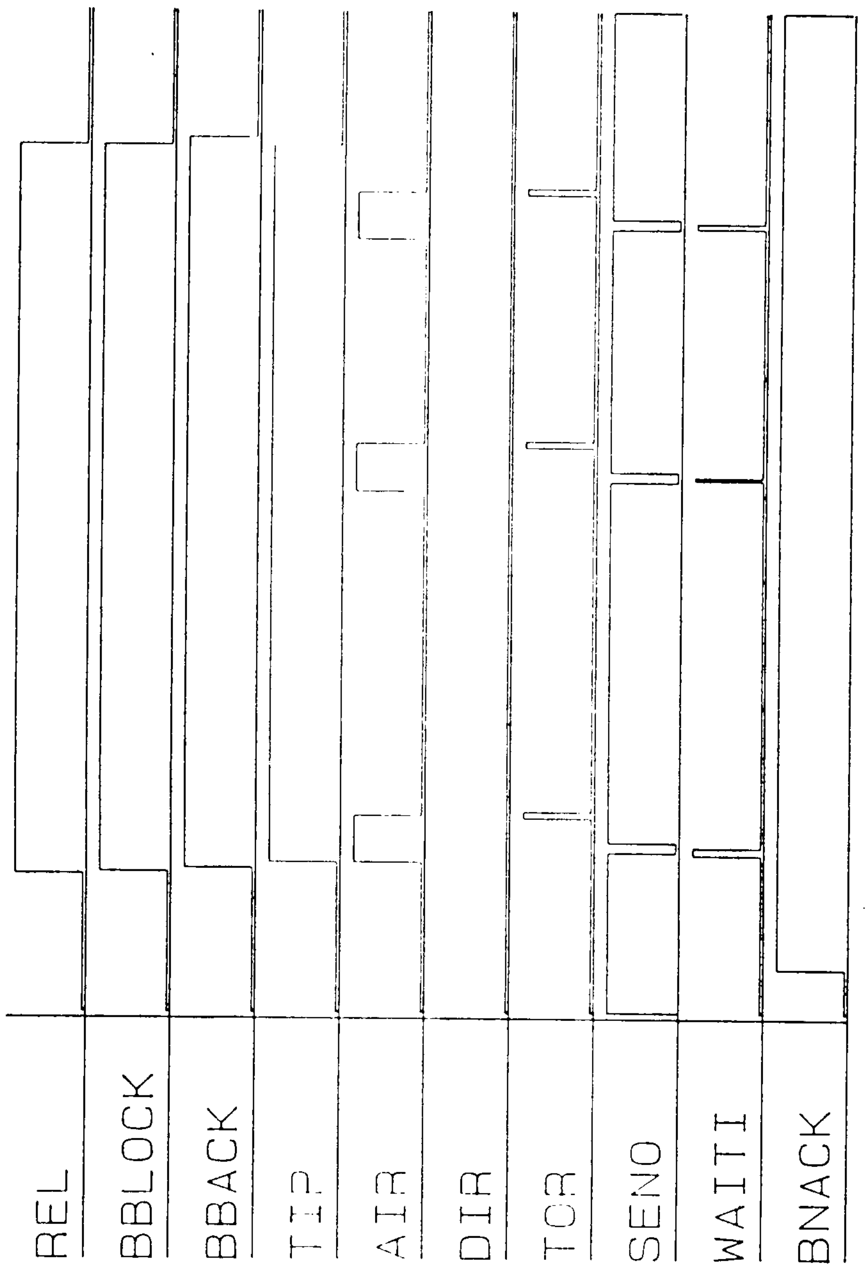


Figure 40. Block read operation for three words (continued)

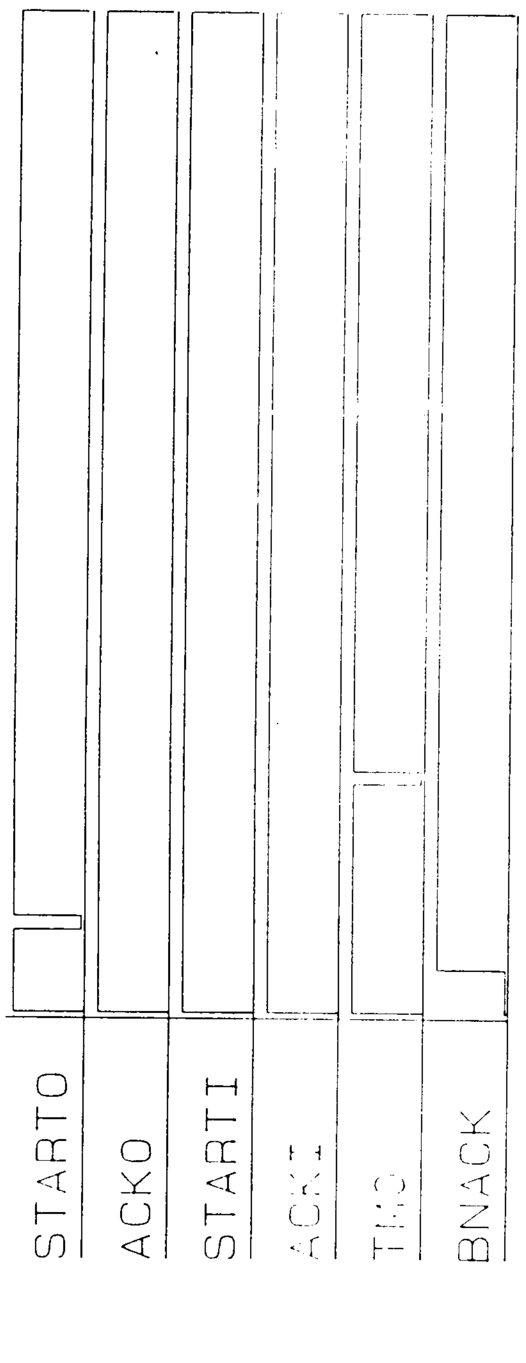


Figure 41. Block read operation for three words (continued)

The signals for the slave BIU are shown in Figure 43. The block start address in incremented by the slave NBC and it is passed to the BTC. (The intermediate addresses are not sent from the master to the slave on the NuFTbus). Figure 42 shows WAITI signal going high till the data read operation on the board is done. SENO remains low till WAITI is high.

Figure 44 shows some of the NuFTbus signals. STARTO goes low for a period of one bus clock width to indicate the start cycle. The intermediate acknowledgments are given on $TM_0$. $TM_0$ goes low to indicate to the master that the last data transfer was carried out successfully. At the end of the block, ACKI goes low to indicate the end of the block write transaction.

Figure 45, Figure 46 and Figure 47 show a block write transaction for a transfer of three words.

The additional results of the simulations carried out by the author are documented in form of a report in reference [20]. Figures in reference [20] show many of the internal signals of the BIU for block read operation. The signals are enable/disable for transceivers, clock and enable inputs to registers and so on. The results shown are for block read transaction for the master as well as for the slave side BIUs. Also, the detailed results for overlapped master and slave operation can be seen in reference [20]. Due to lack of space, detailed results for all transactions are not shown. Note that in all the timing diagrams, the last signal shown is added to match the time scale of all the diagrams of the same transaction.

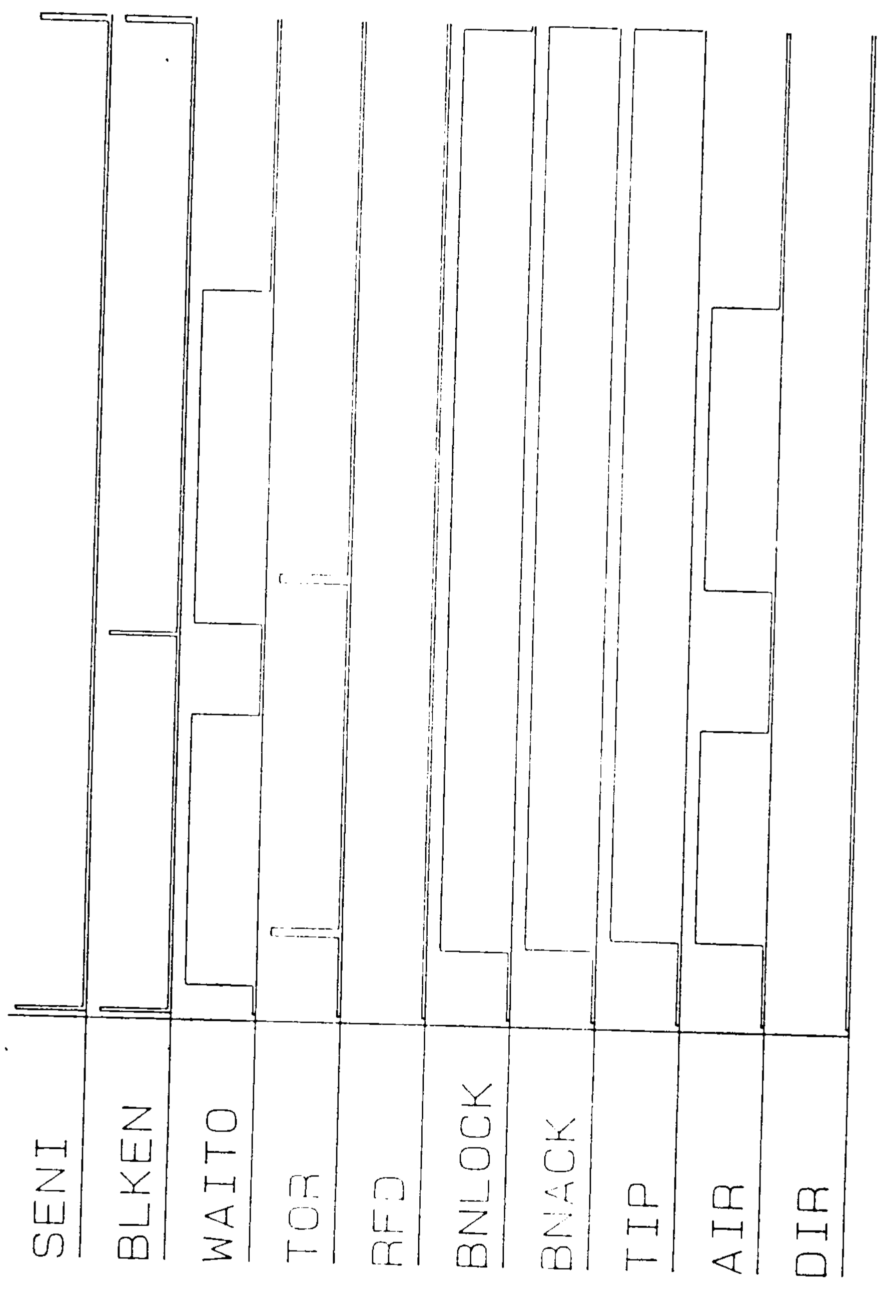


Figure 42. Block write operation

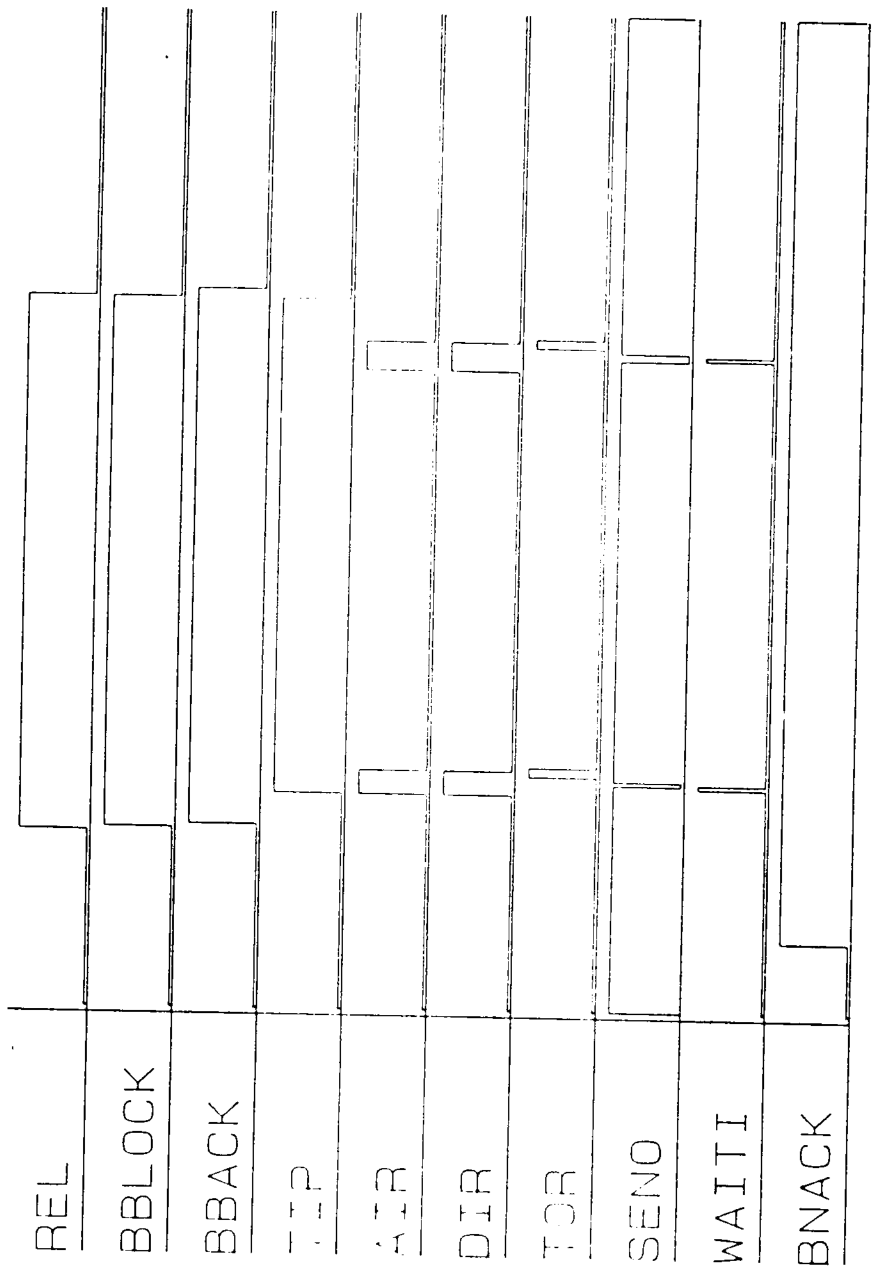


**Figure 43. Block write operation (continued)**

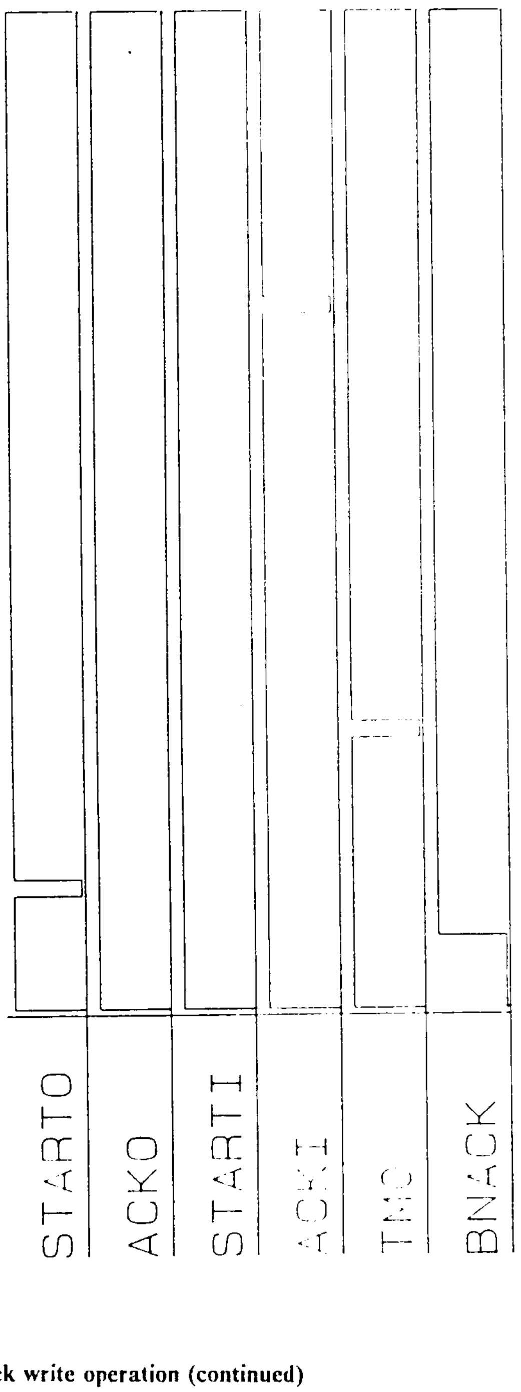


Figure 44. Block write operation (continued)

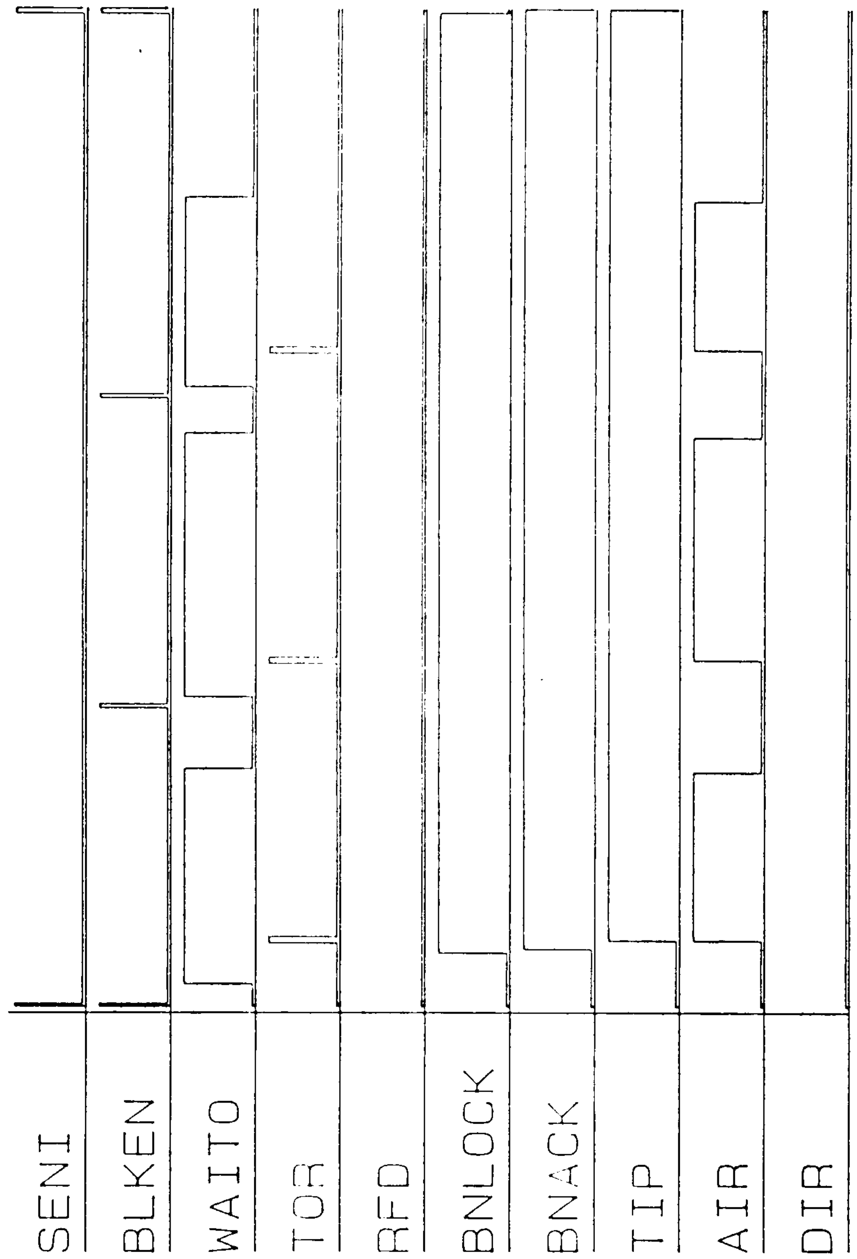


Figure 45. Block write operation for three words

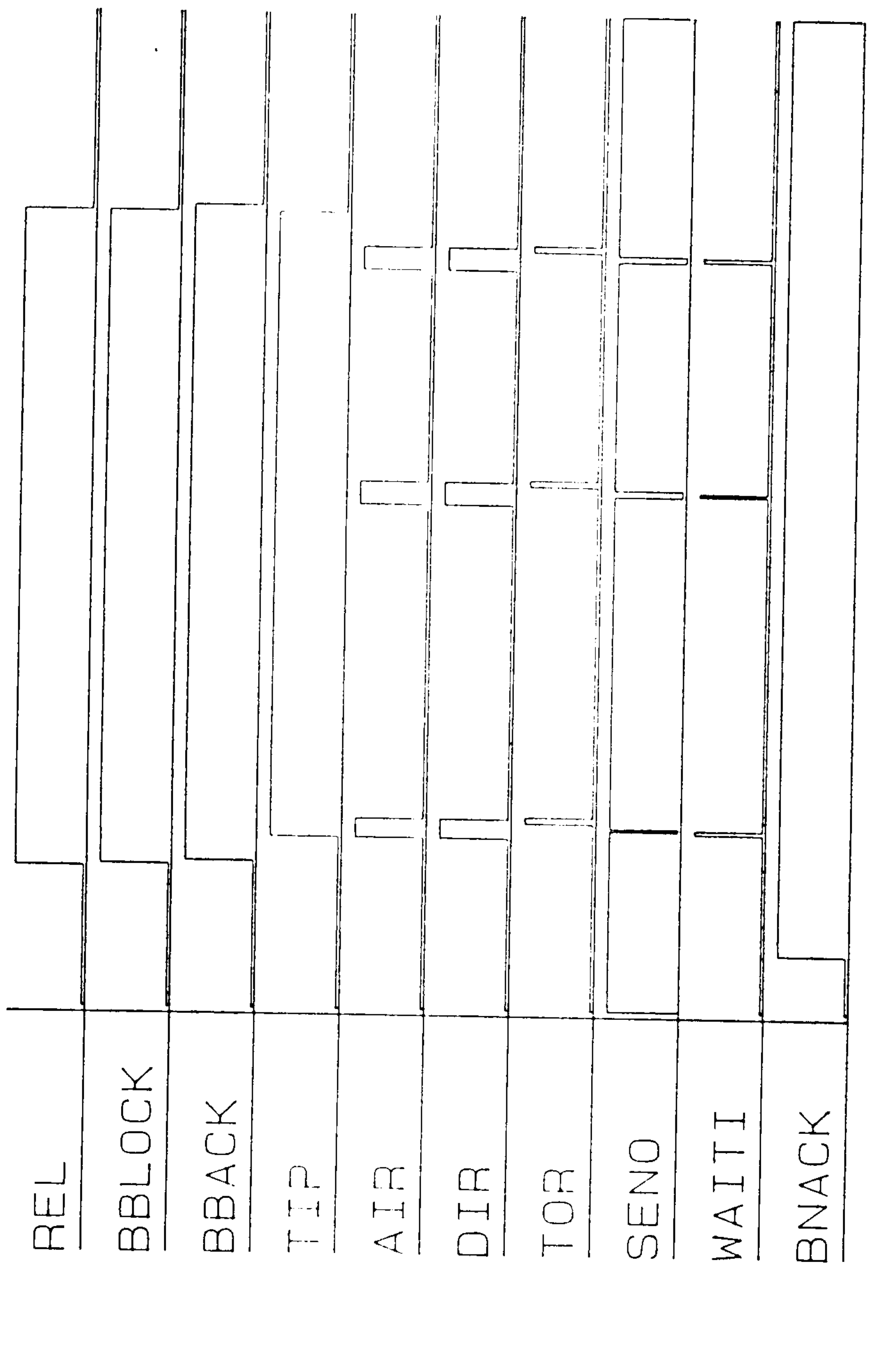


Figure 46. Block write operation for three words (continued)

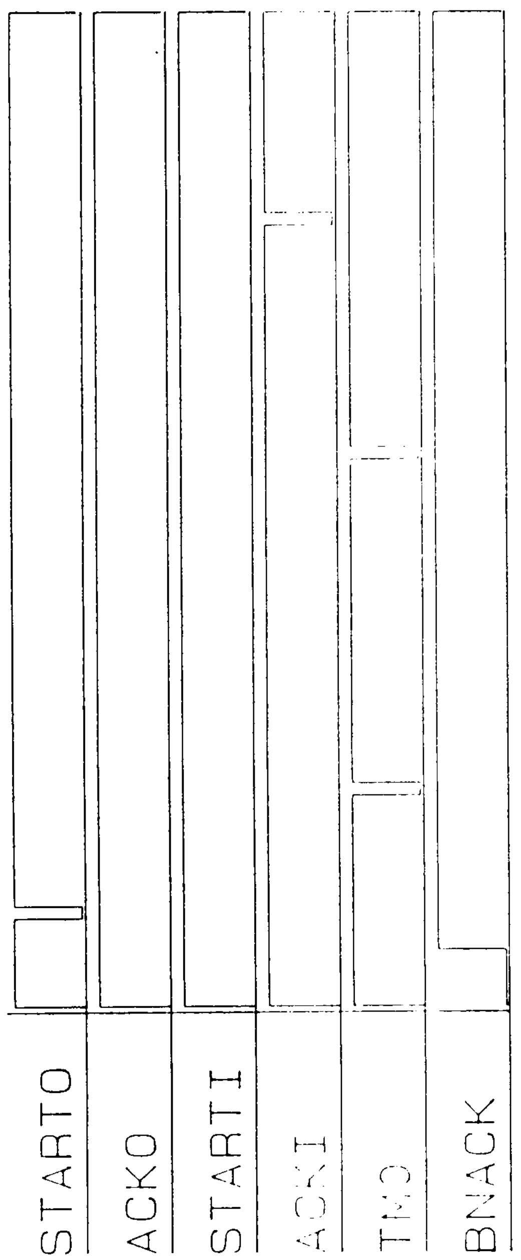


Figure 47. Block write operation for three words (continued)

### 6.2.3 Simulation results of arbitration logic

This section gives the simulation results for the arbitration logic. The simulations are carried out on an IBM-AT based system, P-CAD. The simulator used is PC-LOGS. These simulation results include the fault free behavior of the arbitration logic as well as the functioning in the presence of single bit faults.

**Note :** In the simulation timing diagrams, C : T on the X-axis gives the time scale where C is the cycle number and T is the number of time steps within a cycle. A time step is the smallest unit of the simulation time. It has no particular dimension. In these simulations it is assumed to be in nanoseconds. Cycle is an integer number of time steps [22].

In all the simulations, the cycle is set to 1, so C represents the time in nanoseconds elapsed from the beginning of the simulation. The subcircuits CGIA, ABARB and RQFF are tested individually considering all possible input combinations. The simulation results are shown in Figure 48, Figure 49 and Figure 50. Figure 51 shows the simulation result for the arbitration unit for a module shown in Figure 11. (Also Figure 52 shows an ARBERR line going high due to an error). The operation of the overall arbitration logic is checked by connecting four modules together as shown in Figure 53. Four ID codes are assigned to the modules. The grant signal lines are probed to examine the results of the arbitration operation. The simulation of the arbitration logic is explained using the following examples.

#### *6.2.3.1 Fault free simulation*

1. Four modules (A, B, C and D) arbitrate for becoming the bus master. The ID codes of the modules are:

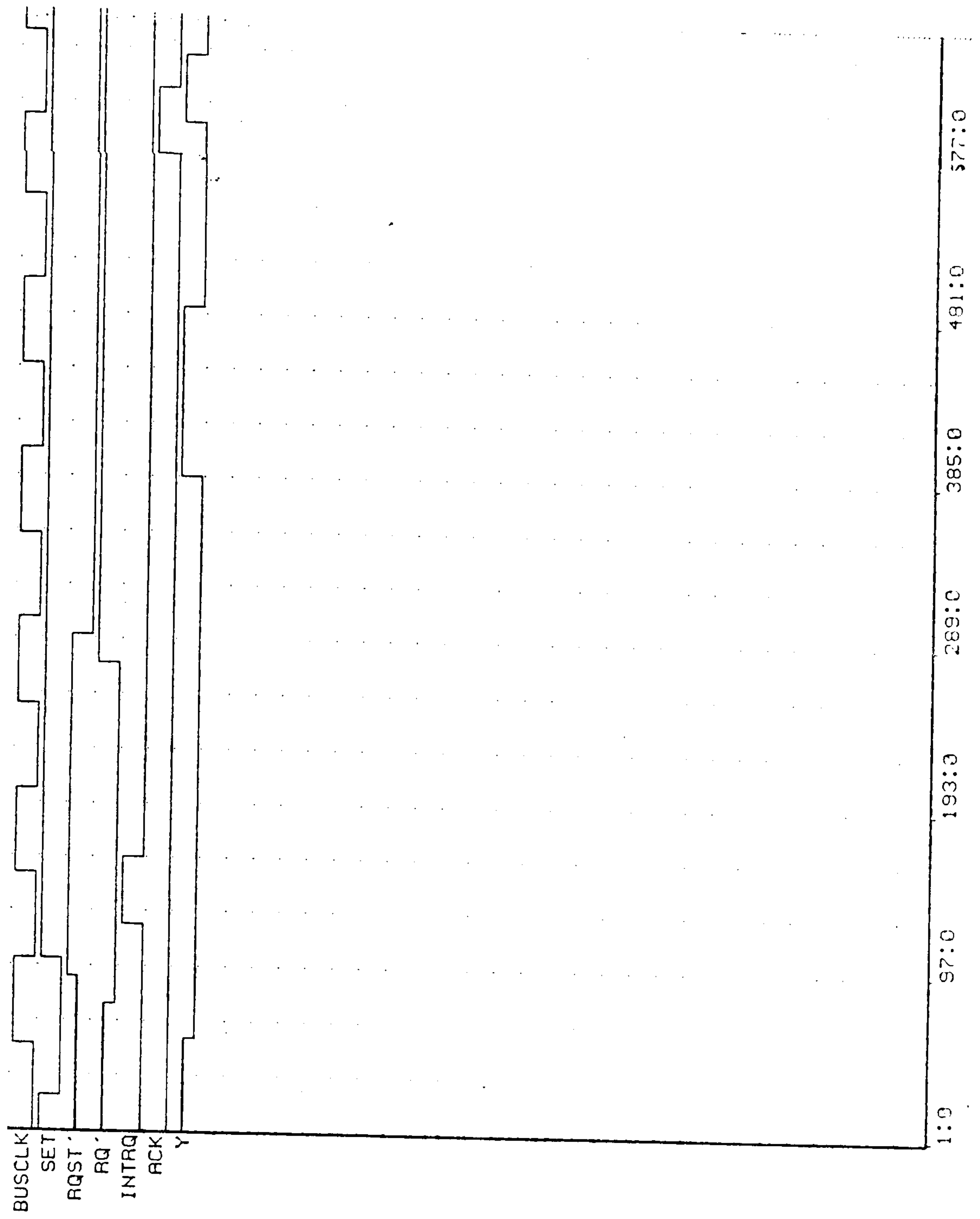


Figure 48. Simulation Result of Subcircuit RQFF.

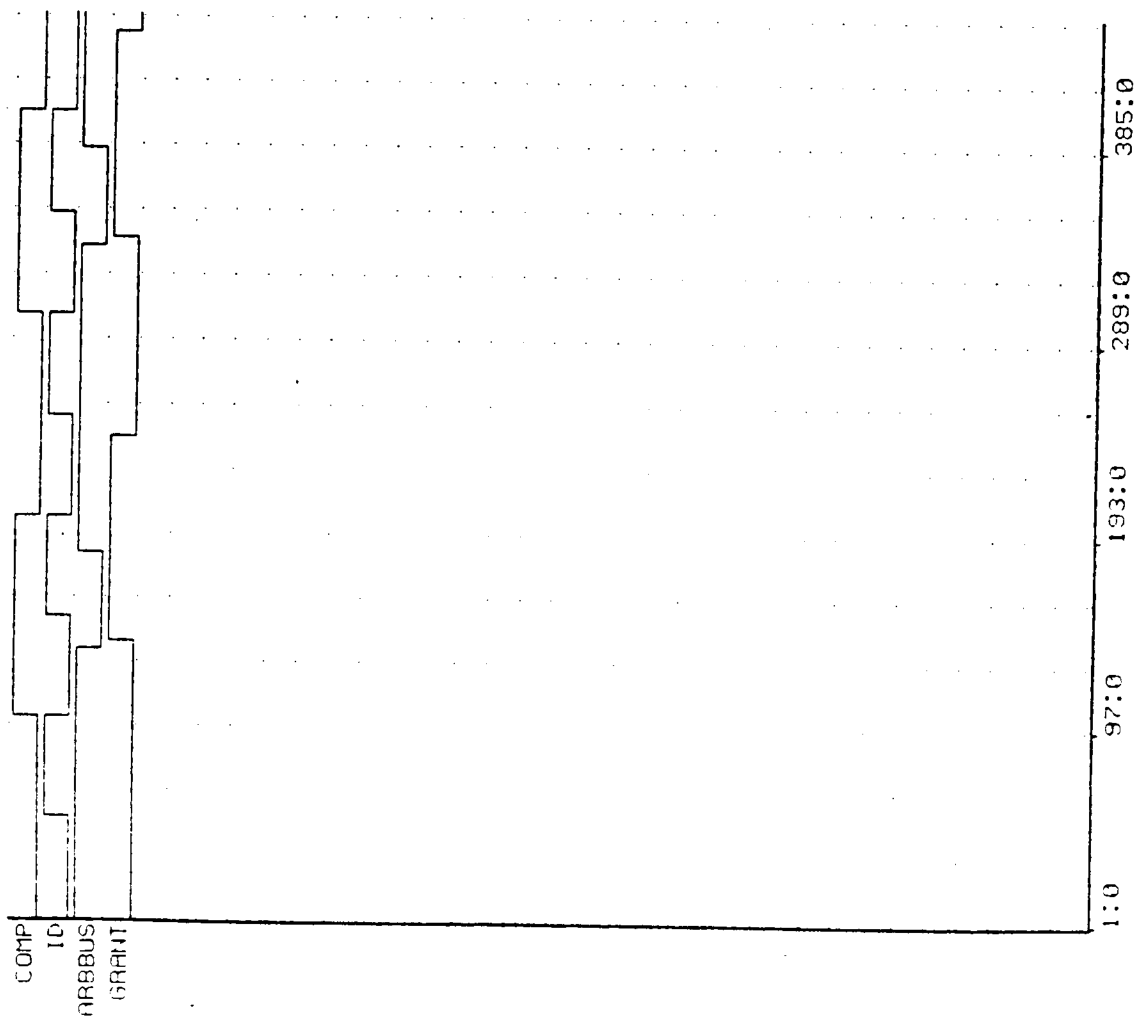


Figure 49. Simulation Result of Subcircuit CGIA.

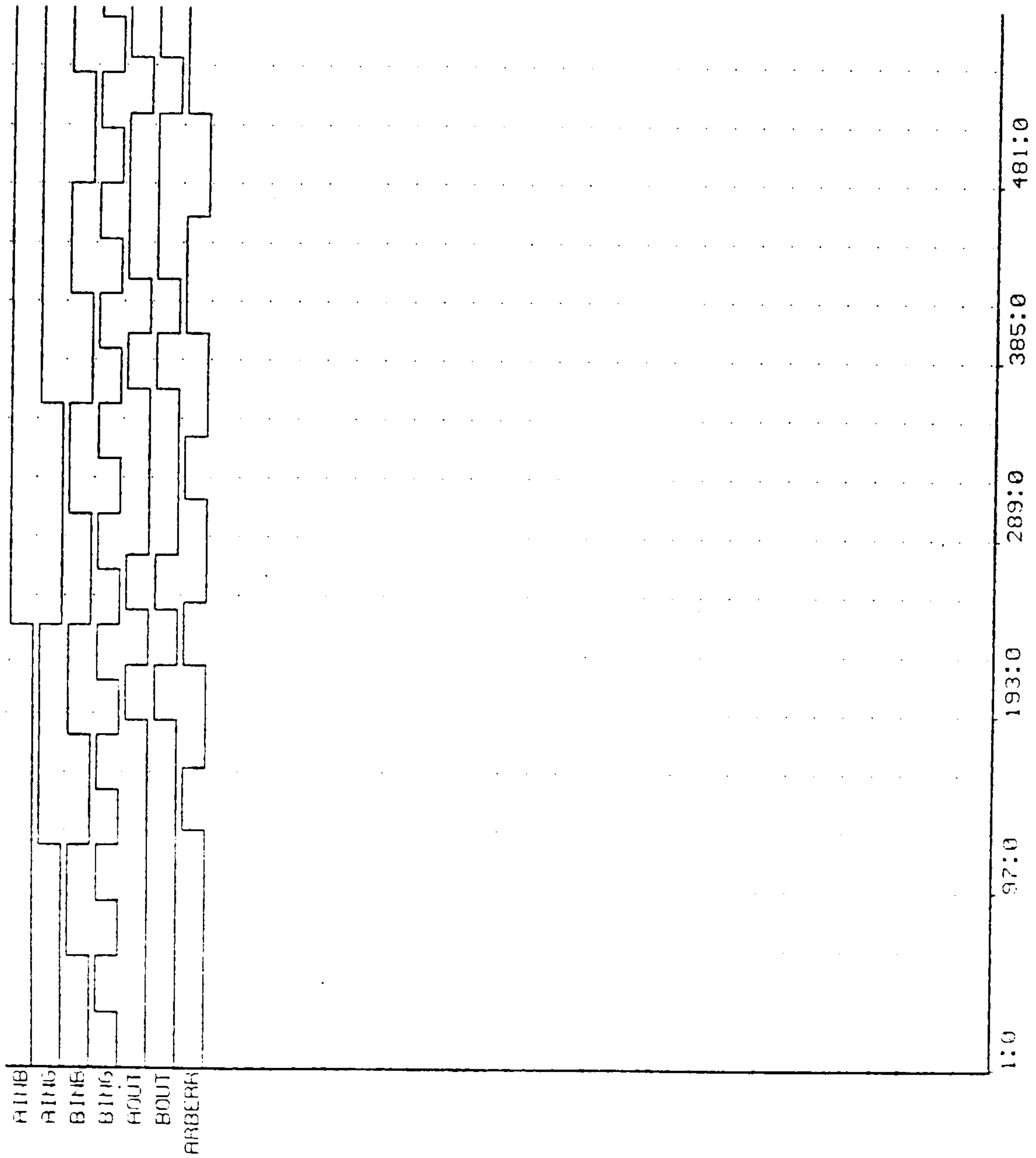


Figure 50. Simulation of Subcircuit ABARB.

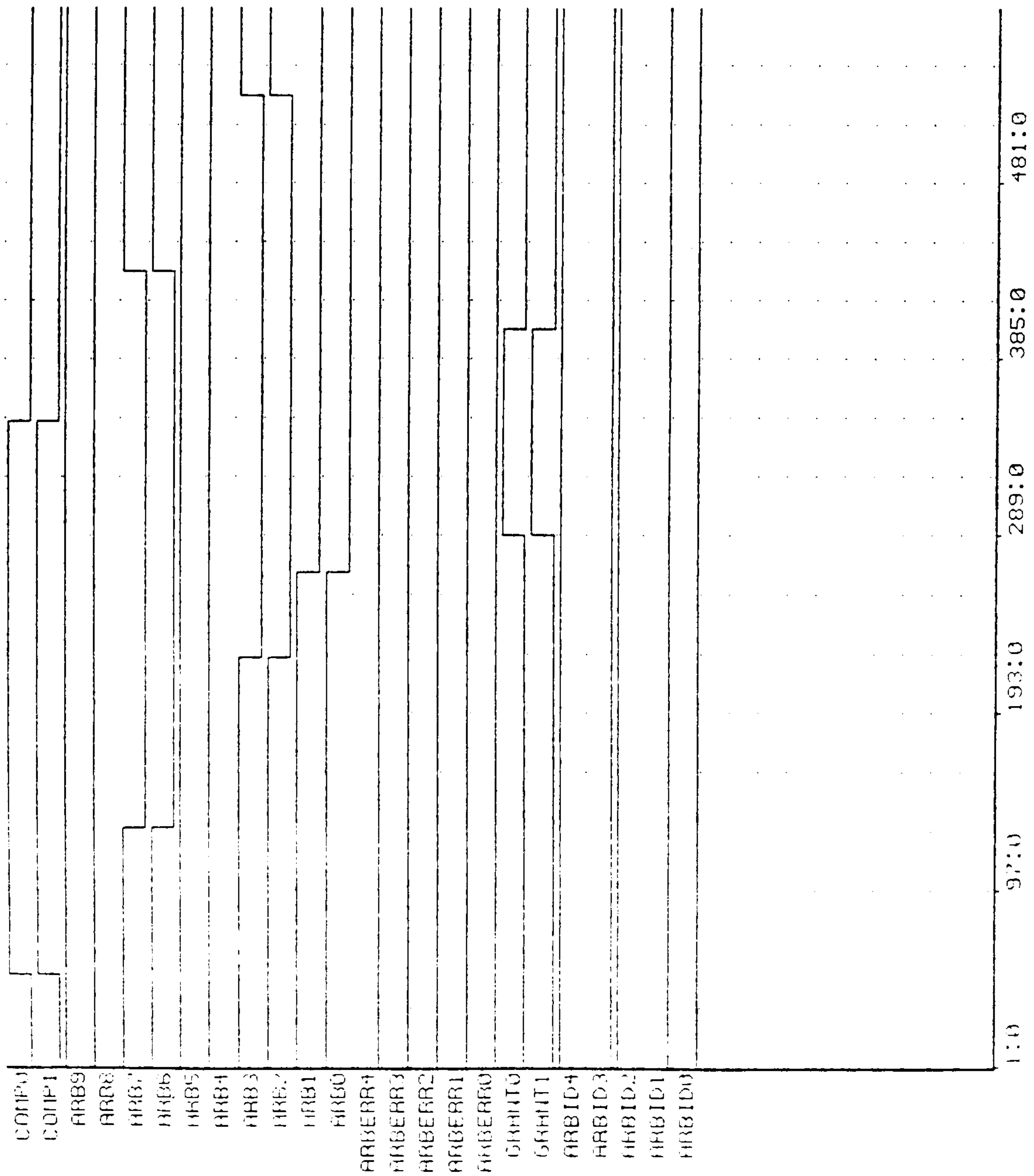


**Figure 51. Simulation Result of Subcircuit ARBI.**

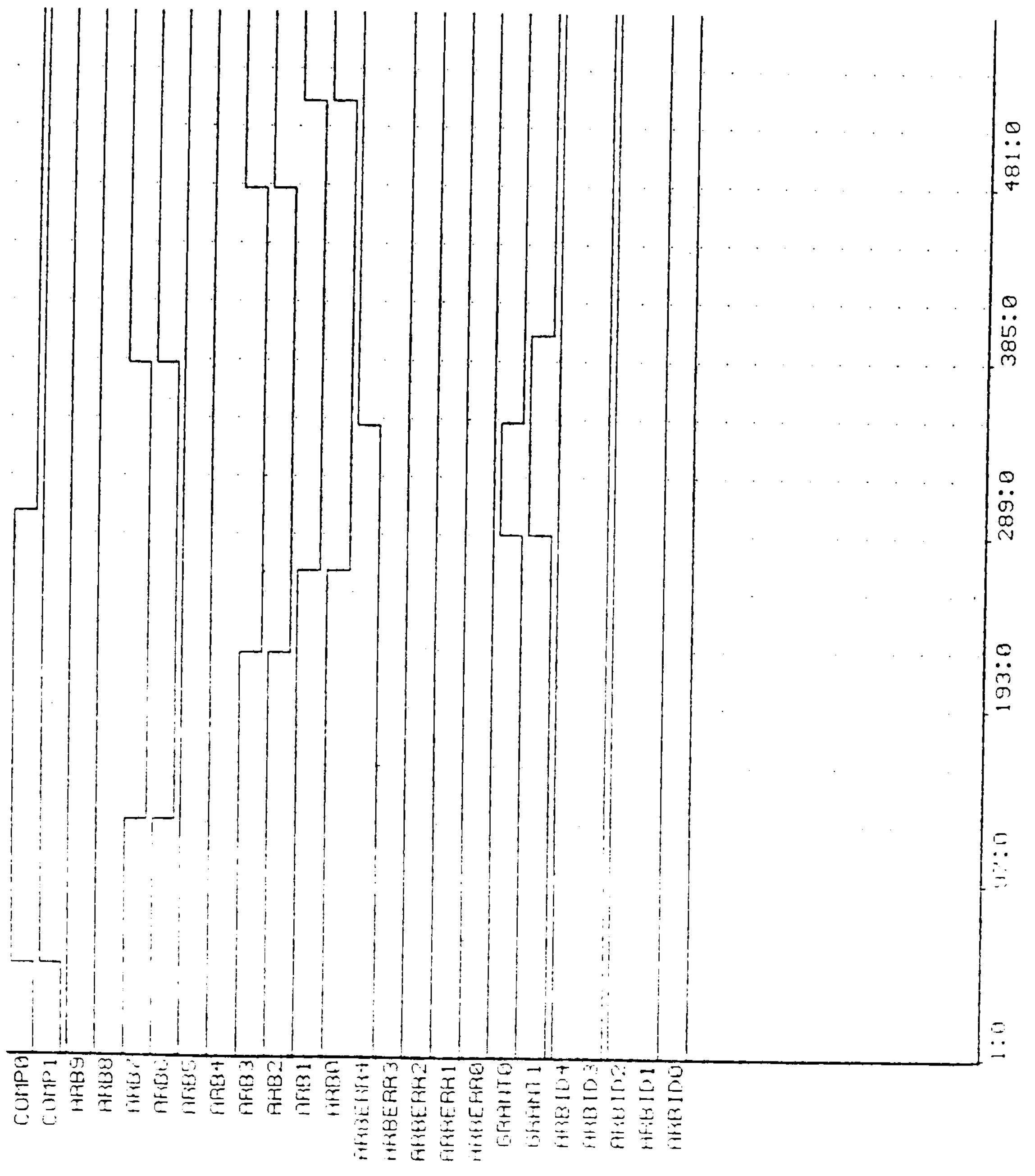


Figure 52. Fault Simulation of Subcircuit ARBI.

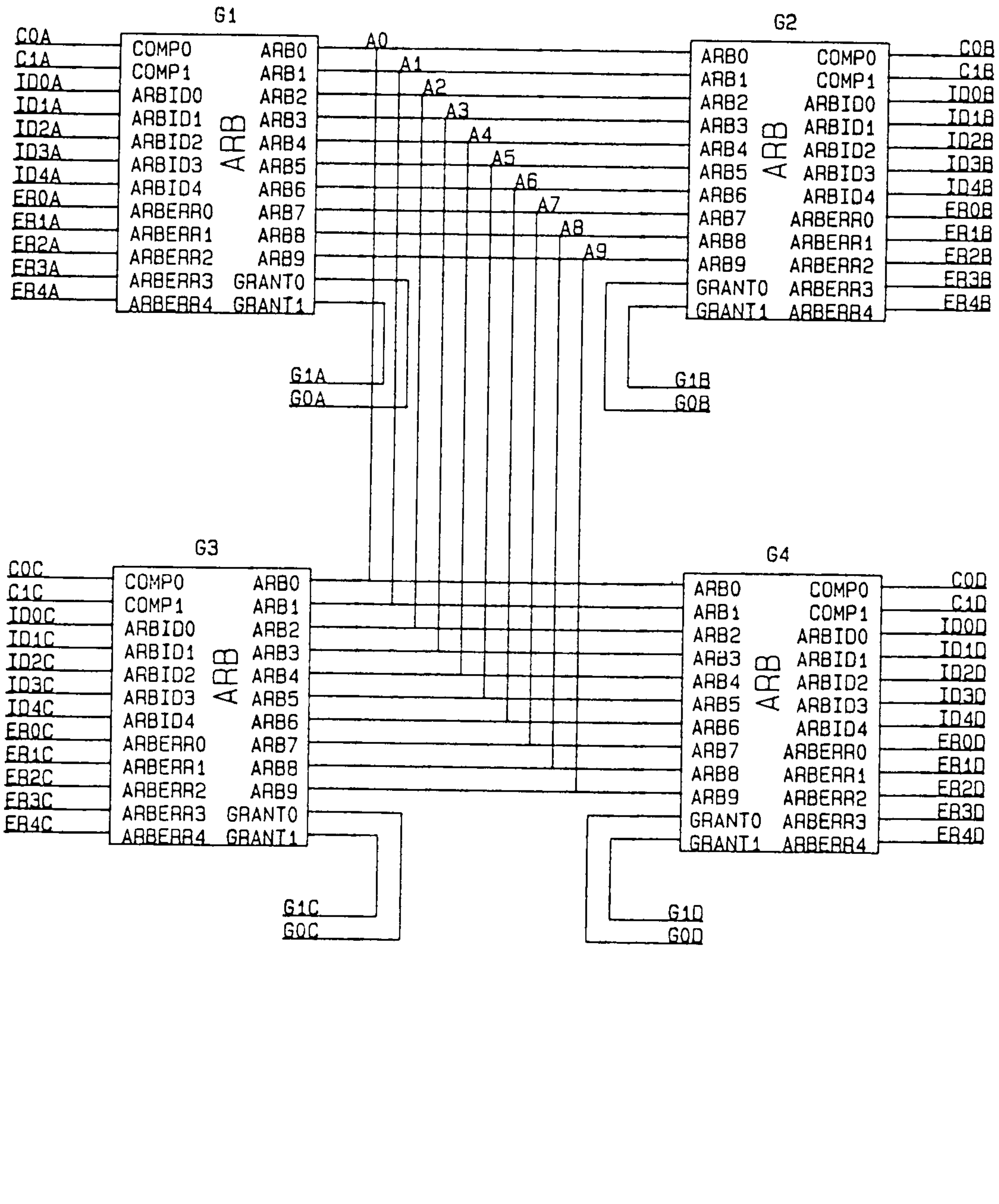


Figure 53. Connection of arbitration units for four modules

A : 10010
B : 11110
C : 11011
D : 00101

Modules A, B and C request the bus by raising the respective Internal Request lines (INTRQA, INTRQB and INTRQC). On the next rising edge of Bus Clock (BCLK), modules A,B and C pull the RQST line low and the arbitration operation starts. At this time, the Internal Request of the module D goes high. But as it misses the clock pulse BCLK which starts the arbitration, the module D has to wait (even if it has the highest priority) till the modules A, B and C get their turns at bus mastership.

The modules A, B and C compete during the arbitration cycle. Since the module A has the lowest ID among the three, it gets control of the bus. ($G_{0A}$ and $G_{1A}$ go high and the ID code of the module A, 10010, is put on the ARB lines). As soon as the module A finishes its transaction, it generates the SETA signal and releases the RQST line. RQST remains low as it is still being pulled down by the modules B and C. With the ACK signal indicating the end of the transaction, the modules A, B and C compete and the module C gets control of the bus. After the module C finishes its transaction the module B alone competes and gets the bus. Once the module B releases the RQST, line RQST goes high (as no module is pulling it down). Now the module D pulls RQST down and enters the arbitration contest. No other module is competing at this time. The module D gets the bus and starts its transaction. Figure 54 shows the timing signals for this arbitration process.

2. Let the ID codes of the four modules be:

   A : 10010
   B : 11110

C : 11011
D : 00101

In this case the modules A, C and D request the bus by raising INTRQA, INTRQB, and INTRQD, respectively. On the next clock pulse, arbitration starts. The module B requests the bus a little later, and has to wait until this round of arbitration is over. The module D gets the bus due to the higher priority of its ID code. After the module D finishes its transaction, the module A and then the module C get the bus. After the module C completes its transaction, a new round of arbitration starts. After finishing its transaction, the module D again requests the bus. Thus, when RQST goes high, the modules B and D request the bus and compete for it. The module D being of higher priority gets the bus, and the module B gets control after the module D finishes its transaction. Figure 55 shows timing signals for this arbitration.

3. Let the ID codes of the four modules be:

   A : 10010
   B : 11110
   C : 11011
   D : 00101

The arbitration operation is similar to that explained in the previous two cases. Initially the modules A and B request the bus. First the module A gets the control of the bus, then the module B gets control of the bus. The modules C and D request the bus later and compete with each other in the next round of arbitration. The module D, having priority higher than that of the module C, gets control of the bus. After the module D finishes its operation, the module C gets the bus. Figure 56 shows timing signals for this arbitration.

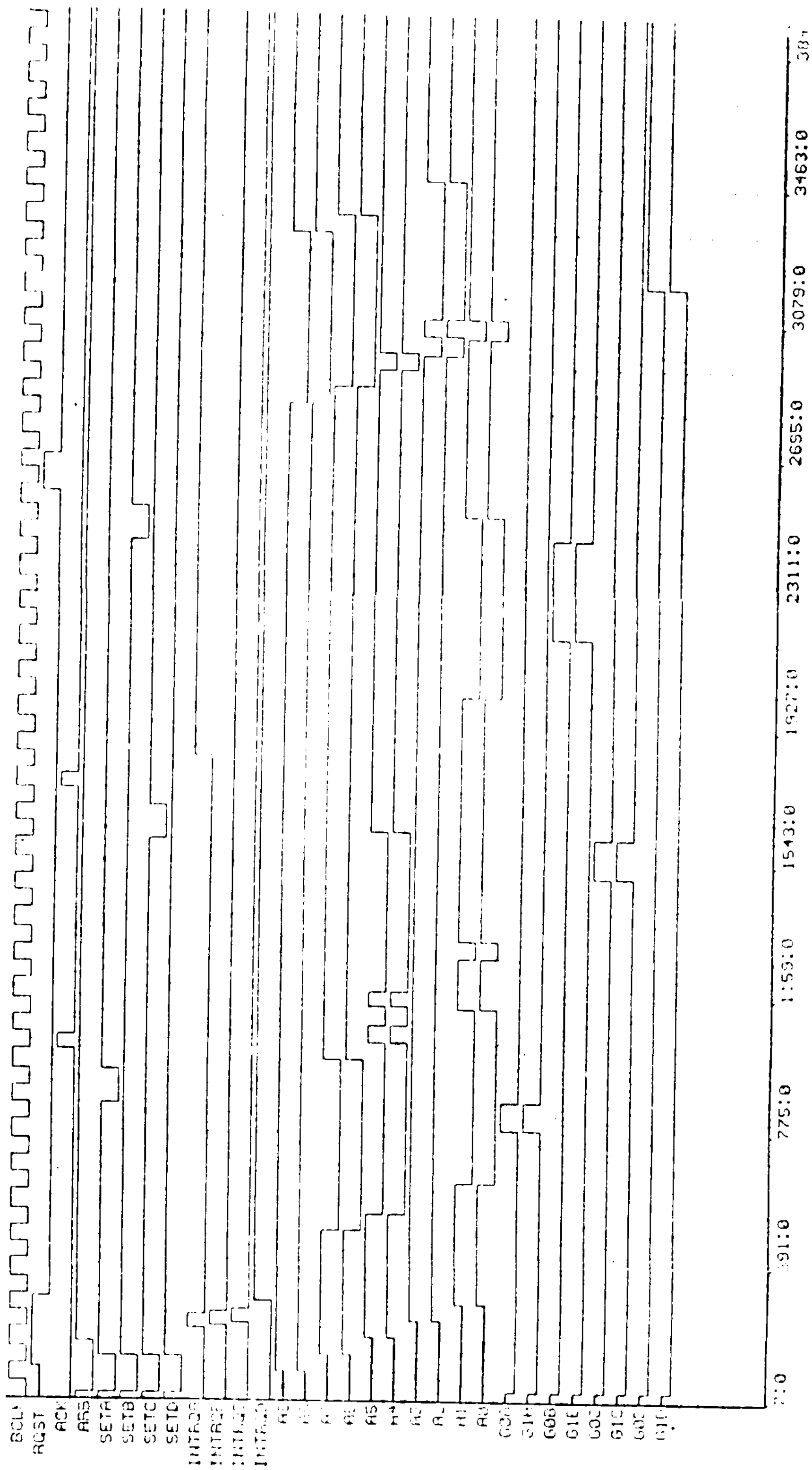


Figure 54. Fault Free Simulation of the Four Module System.

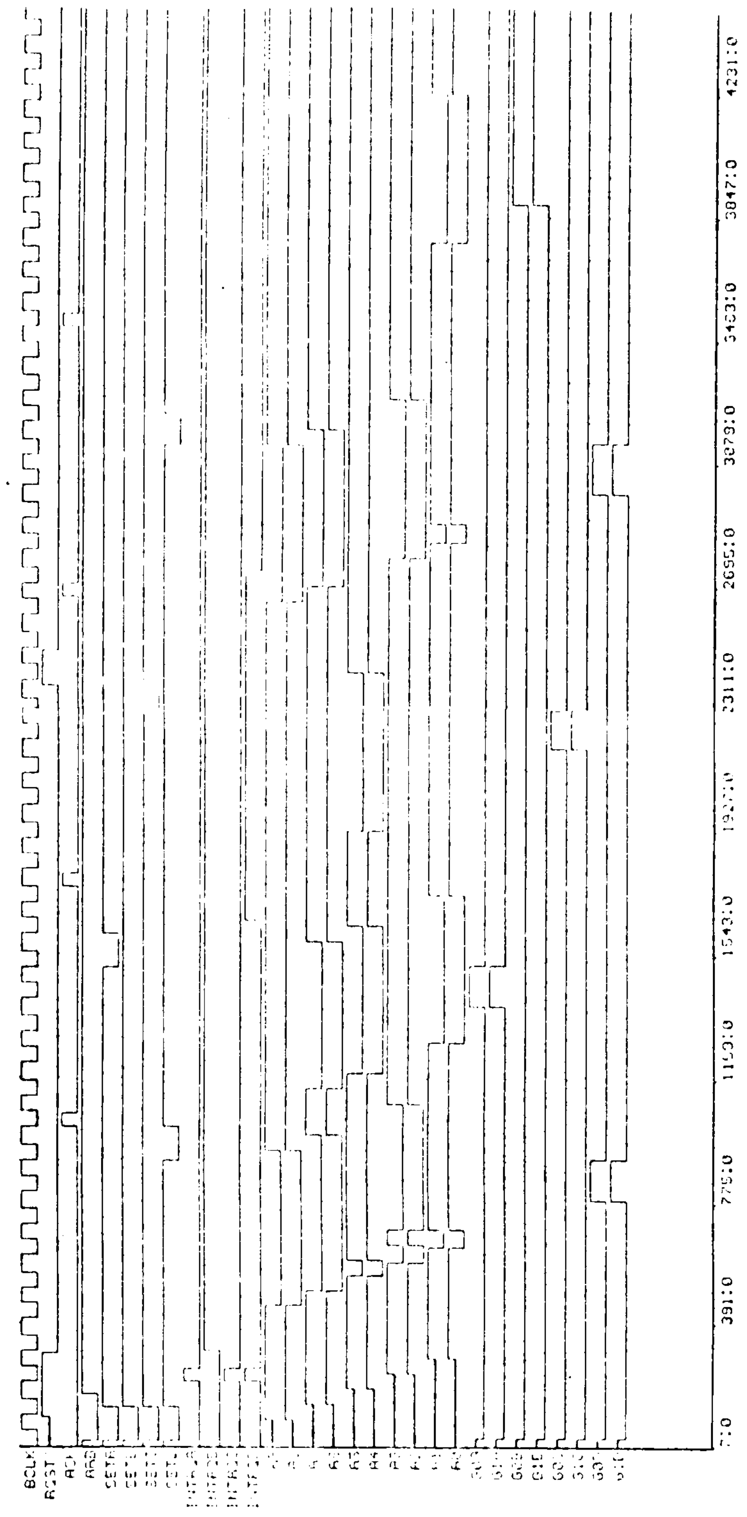


Figure 55. Fault Free Simulation of the Four Module System.

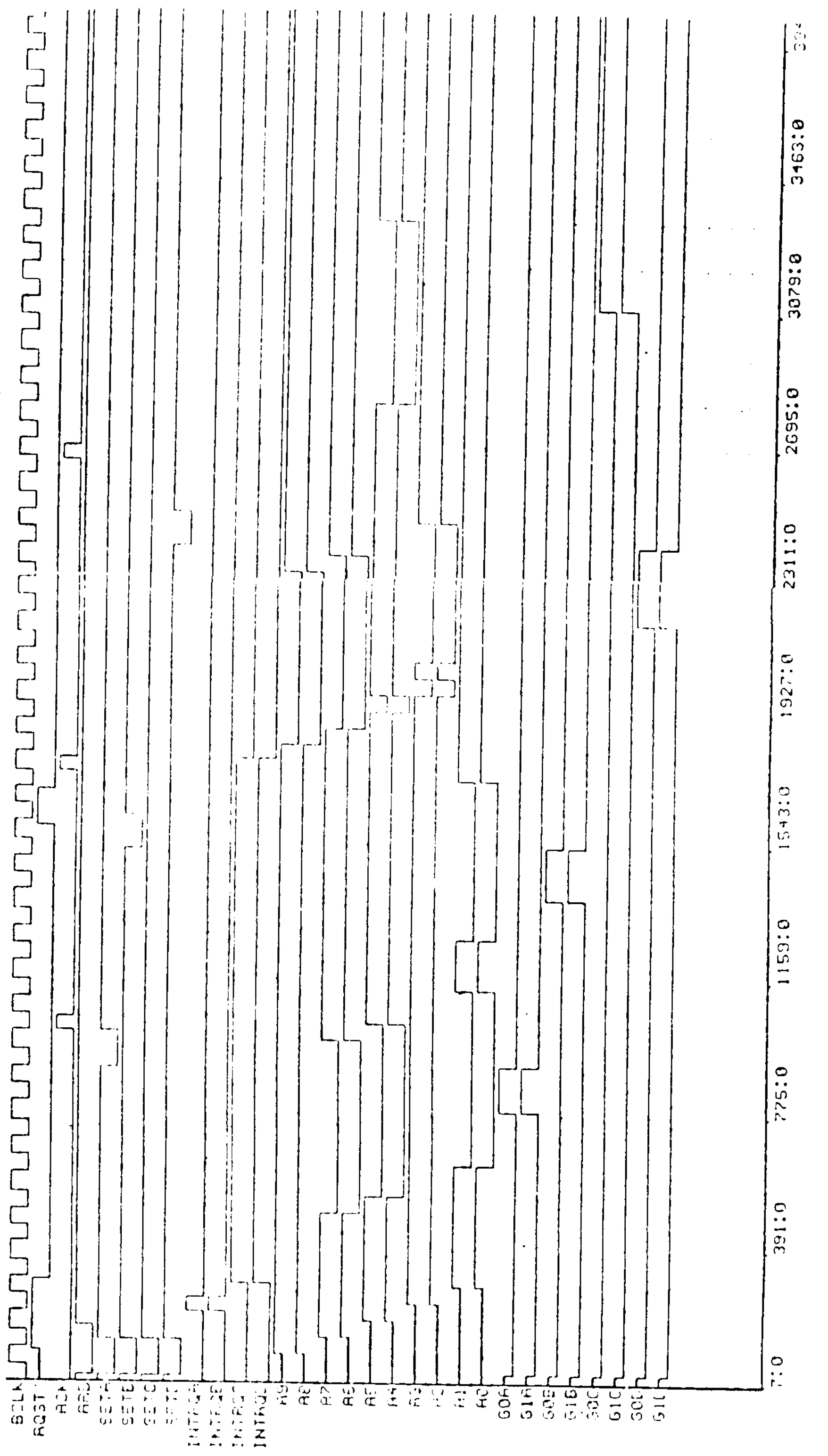


Figure 56. Fault Free Simulation of the Four Module System.

### 6.2.3.2 *Fault Simulation*

For the purpose of fault simulation, it is assumed that only one line (out of 10 lines $ARB_9$-$ARB_0$, two sets of arbitration lines) would be faulty at a given time. A fault is simulated by individually forcing each arbitration line to logic zero or logic one and then carrying out the arbitration. The ID codes of the four modules are:

A : 10010
B : 01010
C : 11011
D : 00101

1. In this example each of the arbitration lines is pulled high (s-a-1) and pulled low (s-a-0) one at a time to simulate the effect of a single bit error in the arbitration logic.

   a. S-A-0 Faults: Consider the case with $ARB_9$ s-a-0. There are two modules with $ID_4 = 0$ in this example. These two modules win the arbitration at $ID_4$ irrespectively of the error and continue the arbitration. The module B loses at $ID_3$ and the module D gets control of the bus in the end. With $ID_3$ s-a-0, the modules B and D win at $ID_4$. The module B loses at $ID_3$ and D wins in the end. Similarly, in all other cases, the modules B and D win the arbitration at the $ID_4$ position and continue bypassing the ID corresponding to the faulty ARB line. In all the cases the module D wins the arbitration. For this situation, the single bit error has no effect on the arbitration, as the module D should have got the control of the bus under fault free conditions since it is the module with the lowest ID. These results can be found in reference [19] in form of a report to document the work done by the author.

b. S-A-1 Faults: Let $ARB_9$ be s-a-1. During arbitration, $ID_4$ is bypassed and the 'modified' ID codes of the modules are:

A : 0010
B : 1010
C : 1011
D : 0101

The 'modified' ID code of A being the lowest, the module A gets the bus. This is also true when $ARB_8$ is s-a-1. When $ARB_7$ (or $ARB_6$) is s-a-1, the $ID_3$ bit is bypassed and the module B gets the control of the bus as it has the lowest 'modified' ID.

A : 1010
B : 0010
C : 1011
D : 0101

In all the other cases the 'modified' ID code of the module D is the lowest and so it gets control of the bus. Thus in all the cases one of the modules gets the control of the bus, even though it might not be the module with the highest priority. All of these results are presented in reference [18].

2. The following example has the same configuration as is shown in Example 1 of the fault free simulation; however $ARB_9$ is s-a-1. The ID codes of the four modules are:

A : 10010
B : 11110
C : 11011
D : 00101

Initially, the modules A, B and C compete. Bit $ID_4$ is bypassed during the arbitration, and the modules A, C and B get the bus in that order. Once the module A finishes its transaction, it again asks for the bus. So now the modules D and A compete for the bus in the second round of arbitration and the module A gets the bus before the module D, as its 'modified ID code (skipping $ID_4$) is less than that of the module D. The module D gets the bus after the module A finishes (Figure 58).

3. In the next example the ID codes are the same as in the above example and again $ARB_9$ is s-a-1. But in this case initially the modules A, B and D ask for the bus. The module A gets the bus first; then the module D and then the module C get the bus. For the second round of arbitration, the module B pulls RQST line low. The module D also requests again and the modules B and D compete. Here, the module D gets the bus before the module B, based on the 'modified' ID codes, and the module B gets control after the module D has finished its transaction (Figure 59).

4. The ID codes for the next example are

   A : 10010
   B : 11110
   C : 11011
   D : 01111

The modules A and B request the bus first and the module A gets the bus followed by the module B. After that, the modules C and D request the bus, and the module C gets the bus before the module D as its 'modified' ID is smaller than that of the module D (the actual ID of the module C is greater than that of the module D). After the module C finishes, the module D gets the bus and completes its transaction (Figure 60).

Thus, it can be seen from all of the above examples that even with a single bit error, the modules arbitrate and only one module gets control of the bus at any time.

The fifth bit in the ID, the parity bit, gives the IDs a distance of two, and so, even if one bit is bypassed, the codes still remain distinct and the arbitration is carried out. In all the cases two modules never win the arbitration at the same time. With a bit error, the modules shift their priorities and get the control of the bus in a different sequence.

To demonstrate the fact that even with a bit error in the ID, the modules will arbitrate fairly, a program model is written in which two modules arbitrate for the bus. Each bit of the ID of one module, module A, is complemented one at a time simulating the effect of a bit error and the module A with this new ID arbitrates with module B. The ID of the module B is changed periodically from 00000 to 11110. These combinations of IDs cover all the possible cases. For example, assume that the ID of the module A is 01010 and with a bit error it becomes 11010. While arbitrating with the module B, when the module B's ID is 11011, the module A gets the bus irrespectively of the bit error module. But with the ID of the module B equal to 10001, the module B gets the bus (without the bit error the module A would have obtained control of the bus).

It is observed that for all the combinations of IDs and error locations, either the module A or B always gets control of the bus, and both of them never get the bus at the same time. The results of the simulations on the different combinations of the modules A and B are presented in the reference [19].

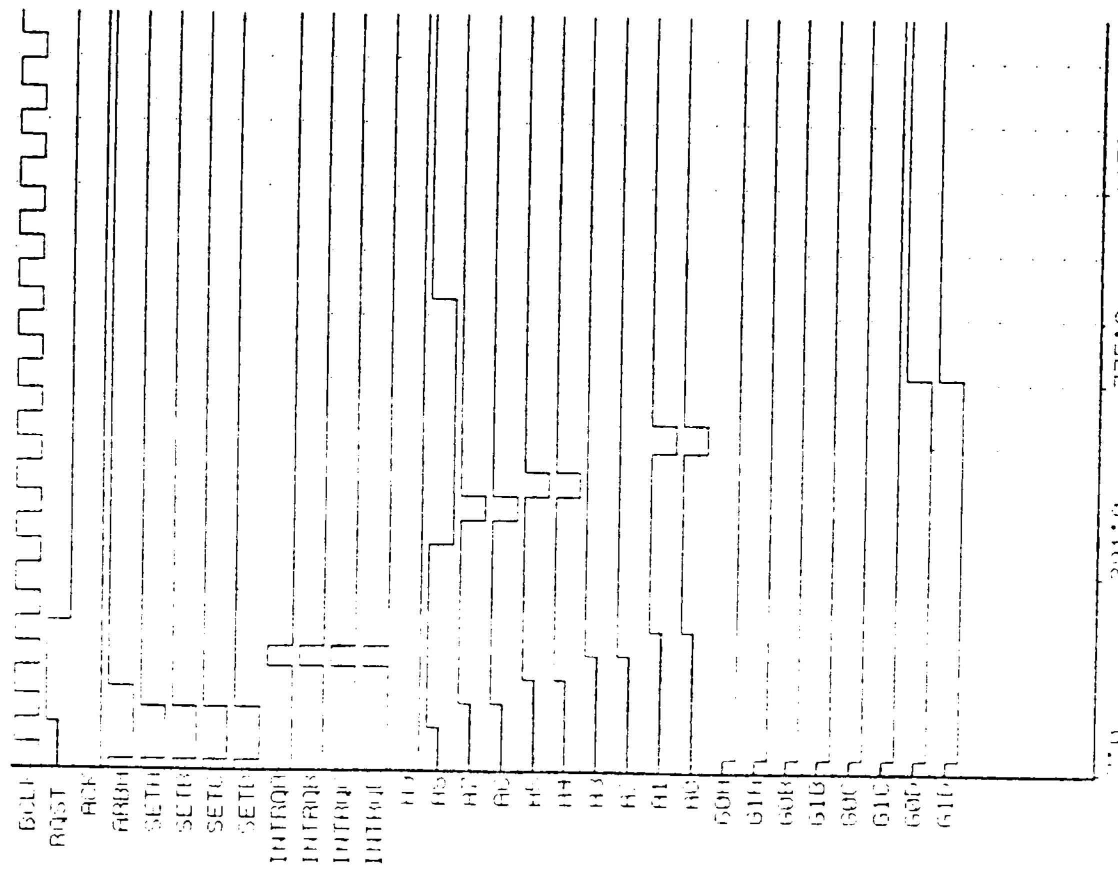


Figure 57. The Fault Simulation of the Four Module System.

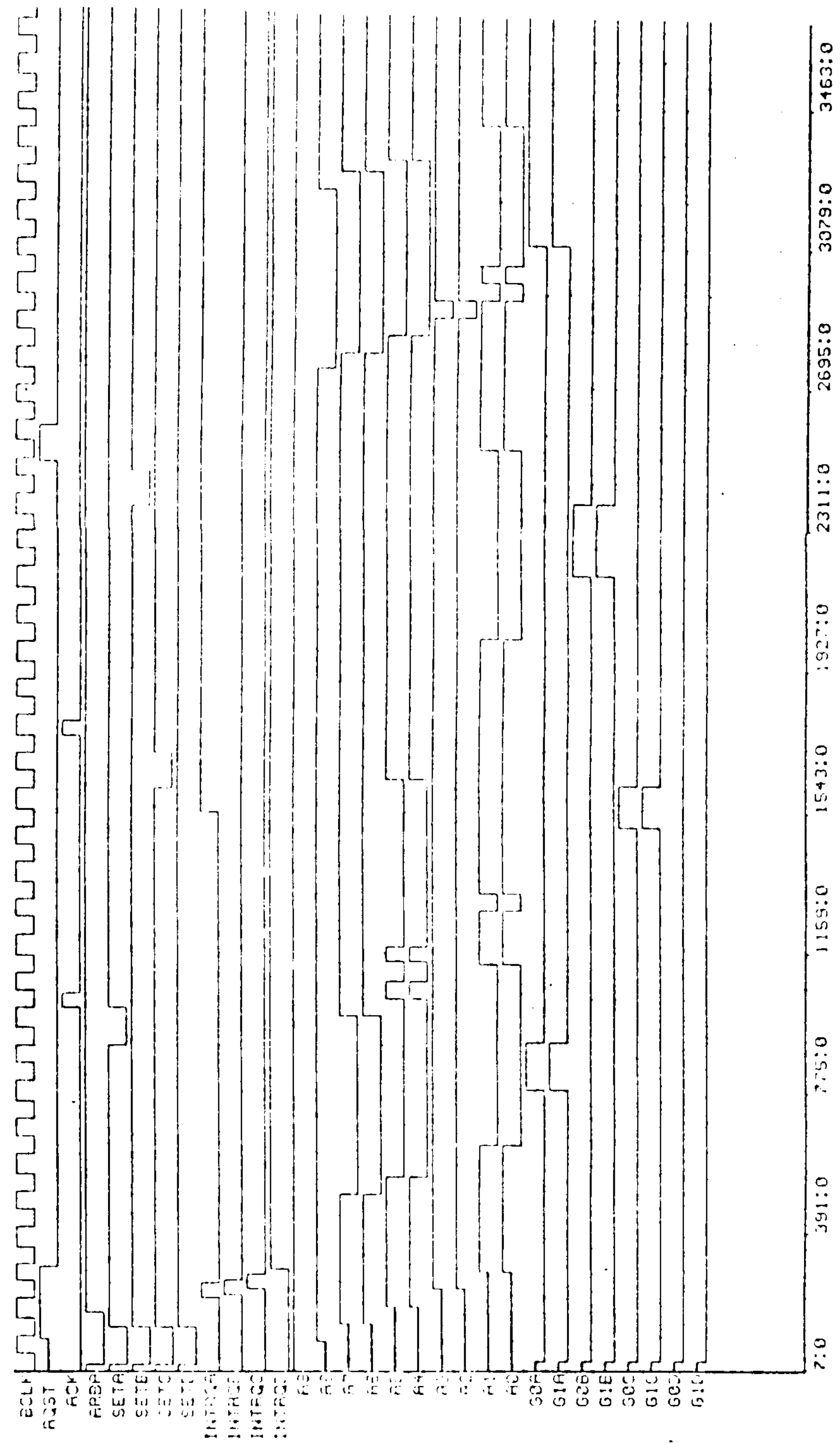


Figure 58. The Fault Simulation of the Four Module System.

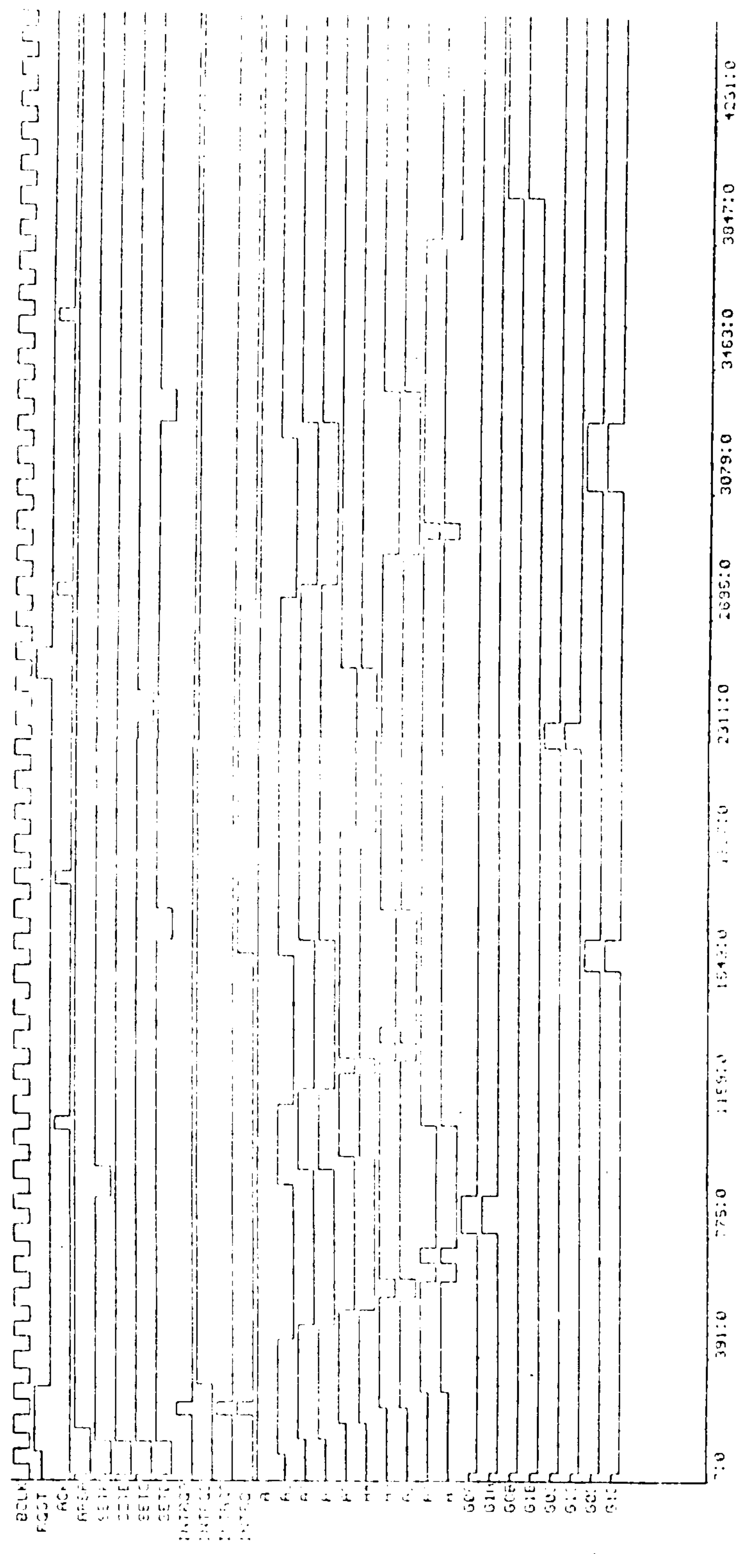


Figure 59. The Fault Simulation of the Four Module System.

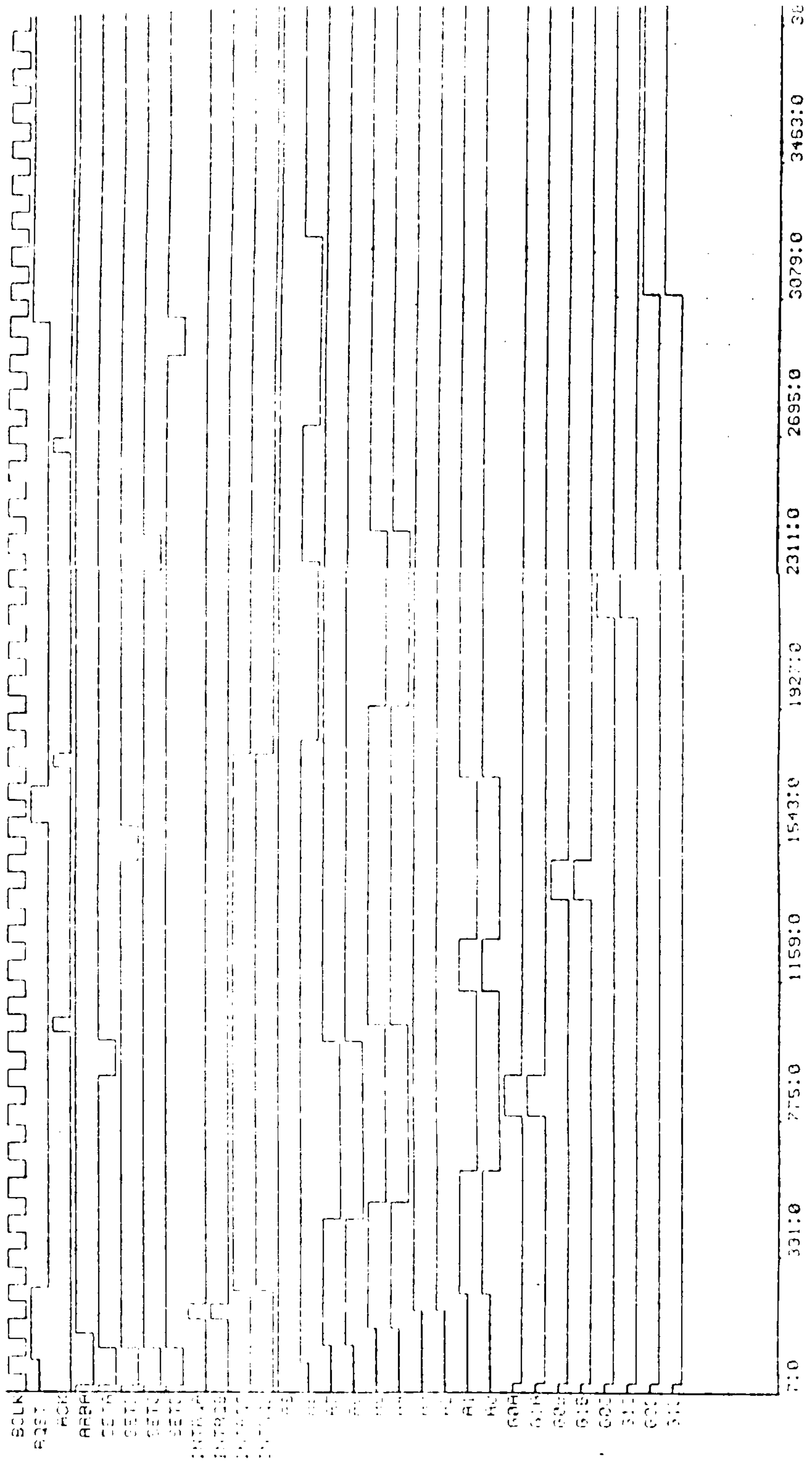


Figure 60. The Fault Simulation of the Four Module System.

# 7.0 IMPLEMENTATION

## 7.1 Introduction

This section describes the implementation procedure for the BIU. The design is implemented as a gate array using the HIGHLAND Design System from the United Technologies Microelectronics Center [23].

The HIGHLAND system accepts input from a schematic capture system working on Daisy, Mentor or Valid workstations. It can also accept inputs in the form of net lists. For implementing, the design is divided into two parts. All the combinational logic circuits, such as error detection and correction units, arbitration unit, address decode units, are entered in the form of net lists. Also, the data path consisting of different registers and transceivers is entered using net lists. For entering the two control units NBC and BTC, a different approach is taken. Berkeley tools (CAD tools developed at University of California, Berkeley) [24] are used to enter the NBC and the BTC. The BTC and the NBC are entered in a finite state machine compiler, PEG, whose output is channelled through the other tools, EQNTOTT and ESPRESSO. The output of ESPRESSO is passed to the HIGHLAND system.

## 7.2 Net lists

HIGHLAND system uses McLDL (Microelectronics Center Logic Description Language) for entering the net lists. Details of the syntax for McLDL are in the HIGHLAND Reference Manual [23]. HIGHLAND system provides a cell library which consists of logic cells like AND2, NAND4, buffers, storage cells, I/O cells, and macros for connecting to the output pads. HIGHLAND system allows users to define their own macromodules which can be used in a hierarchical design.

Different macromodules, such as 40-bit register, 4-bit synchronous counter, comparator and incrementer, are defined and used in the design hierarchy. This simplifies the task of connecting large circuit blocks. For example, Figure 61 shows a macromodule R8OE. It is an 8-bit register with an output enable. On the rising edge of the clock input C, the data D0-D7 gets latched in the register. When EN goes high, the latched data is passed to the outputs O0-O7. The outputs are tristated when EN is low. Figure 62 shows an McLDL description for a 40-bit register, designed using the macromodule R8OE.

Net lists for different components of the design are entered, and then all of the components are interconnected. The major components are 32-bit error detection and correction unit, address decode unit, register array, the NuFTbus transceiver and latch, and a 32-bit address incrementer. The following section describes design considerations for some of the components.

```
MODULE R8OE

INPUT  ENABLE C D0 D1 D2 D3 D4 D5 D6 D7

OUTPUT        O0 O1 O2 O3 O4 O5 O6 O7

NETWORK

N8    INV1   C
CLK    INV1   N8
N9    INV1   ENABLE
EN    INV2   N9

G0    DFF    CLK D0 : N0 @
G1    DFF    CLK D1 : N1 @
G2    DFF    CLK D2 : N2 @
G3    DFF    CLK D3 : N3 @
G4    DFF    CLK D4 : N4 @
G5    DFF    CLK D5 : N5 @
G6    DFF    CLK D6 : N6 @
G7    DFF    CLK D7 : N7 @

M0    TS2   EN   N0 : O0
M1    TS2   EN   N1 : O1
M2    TS2   EN   N2 : O2
M3    TS2   EN   N3 : O3
M4    TS2   EN   N4 : O4
M5    TS2   EN   N5 : O5
M6    TS2   EN   N6 : O6
M7    TS2   EN   N7 : O7

ENDMODULE
```

**Figure 61. McLDL description for an 8-bit register**

```
MODULE R40OE

INPUT  ENABLE C D0 D1 D2 D3 D4 D5 D6 D7          +
           D8 D9 D10 D11 D12 D13 D14 D15     +
           D16 D17 D18 D19 D20 D21 D22 D23  +
           D24 D25 D26 D27 D28 D29 D30 D31  +
           D32 D33 D34 D35 D36 D37 D38 D39

OUTPUT        O0 O1 O2 O3 O4 O5 O6 O7          +
           O8 O9 O10 O11 O12 O13 O14 O15     +
           O16 O17 O18 O19 O20 O21 O22 O23  +
           O24 O25 O26 O27 O28 O29 O30 O31  +
           O32 O33 O34 O35 O36 O37 O38 O39

NETWORK

G0    R8OE   ENABLE C D0 D1 D2 D3 D4 D5 D6 D7 :          +
               O0 O1 O2 O3 O4 O5 O6 O7
G1    R8OE   ENABLE C D8 D9 D10 D11 D12 D13 D14 D15 :    +
               O8 O9 O10 O11 O12 O13 O14 O15
G2    R8OE   ENABLE C D16 D17 D18 D19 D20 D21 D22 D23 :  +
               O16 O17 O18 O19 O20 O21 O22 O23
G3    R8OE   ENABLE C D24 D25 D26 D27 D28 D29 D30 D31 :  +
               O24 O25 O26 O27 O28 O29 O30 O31
G4    R8OE   ENABLE C D32 D33 D34 D35 D36 D37 D38 D39 :  +
               O32 O33 O34 O35 O36 O37 O38 O39

ENDMODULE
```

**Figure 62. McLDL description for a 40-bit register**

### 7.2.1 The Data path

The internal data path is multiplexed with the address path, and both controllers NBC and BTC, use the same data path. Thus, the implementation of the data path is one of the important tasks of the connection procedure. Figure 63 shows a block diagram of the internal data path. As the address and data buses are demultiplexed on the board, two board transceivers are used. In the case of a master write transaction, both the address and data are present at the same time. Similarly, in the case of a slave write operation, both the address and data should be put on the board-BIU interface at the same time. These situations require the presence of more than one word on the internal bus simultaneously. To incorporate this feature, two tristate isolators are used. One isolator is bidirectional, it is enabled and disabled by the signal SEP2 and its direction can be changed by DRSEP2. The other isolator is unidirectional and is controlled by the signal SEP1. The isolators are designed using the internal tristate buffers, TS2, provided by the HIGHLAND cell library. When the signals SEP1 and SEP2 are disabled, the data path is effectively broken up into two parts and then both the address and data can be present at the same time. This is explained by an example. Consider the master write operation. Initially, both SEP1 and SEP2 are disabled, and the address and data from the board-BIU interface are latched into the respective registers (Address is latched in A OT RG (Addr. Out Register) and data is latched in D OT RG (Data Out Register)). To generate/check EDAC on the address, SEP1 is enabled (SEP2 is disabled). This connects A OT RG to EDAC unit. The address is put on the internal bus by enabling AOROE. The output of the EDAC is enabled using EDACOE and the modified word is stored in A OT RG using LAO signal. Then SEP1 is disabled.

The EDAC is generated/checked on the data word in the same way, except that SEP2 is enabled instead of SEP1.

In the slave mode, when the NBC detects a START cycle, it has to latch the address on the NuFTbus immediately. Address lasts on the bus only for the duration of the START cycle, i.e., 100 ns. Because this duration is not sufficient for NBC to determine if the data path is available or not, a 40-bit latch is provided in the NuFTbus transceiver (TRANSL). The incoming address is stored in this latch. Once NBC has the internal bus, the address is passed to the Address In Register (A IN RG) by enabling SEP2 and changing the direction of the isolator by DRSEP2.

The data path is used in a similar way for other operations of the BIU.

### 7.2.2 Control path

Figure 64 shows the control path in the BIU which is connected to the control lines on the NuFTbus. The control lines are $TM_1$, $TM_0$, START and ACK ($AD_1$ and $AD_0$ are also used to encode the type of operation). The control path consists of a 4-bit EDAC unit (for generating three check bits and a parity bit), an 8-bit register with output enable (for storing incoming control/status signals from the NuFTbus), a 4-bit register with output enable (for storing control signals from the board), a 2-bit latch (for generating the acknowledgement) and two 3 to 8 decoders. The decoders are always enabled. Note that the registers with output enable have two separate parts, a register and a tristate buffer. The inputs to the decoders are provided from the output of the register before tristating. This ensures that the control lines are always available in the decoded state, and moreover, the control path can be tristated whenever necessary.

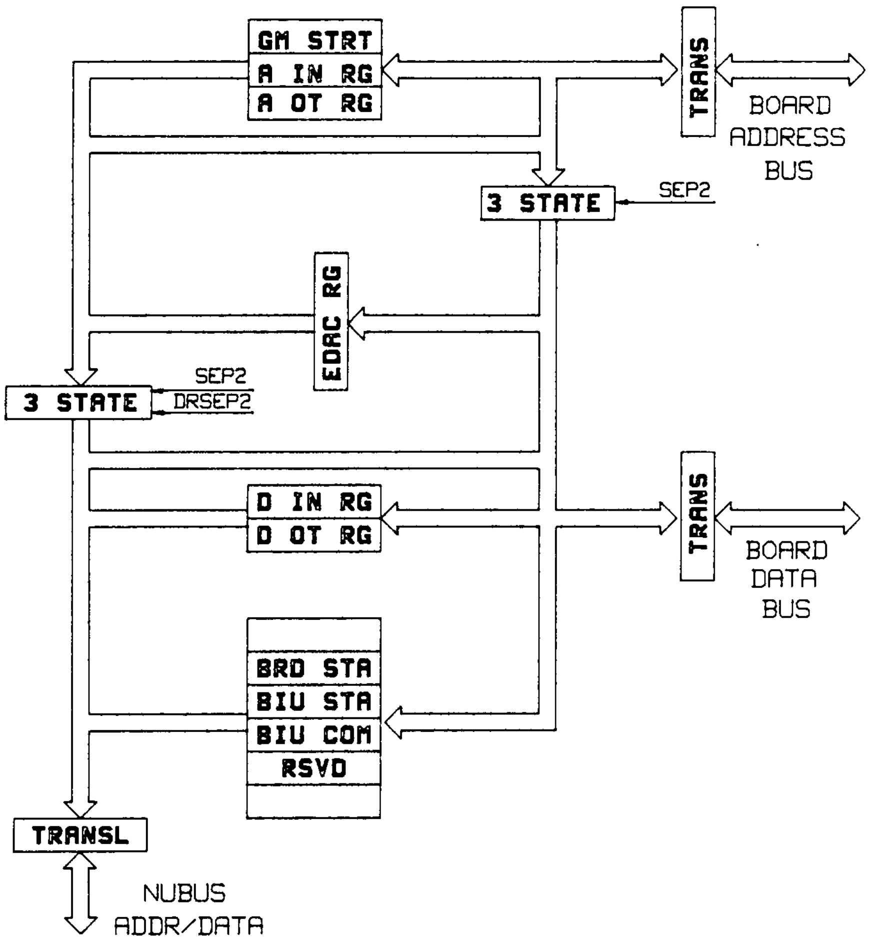


Figure 63. Internal data path

In the case of the master read/write operation, the signals from the board are latched by BTC in the 4-bit register using LWSO signal (Latch Word Size Out). The control lines consist of RWBO, B/O*, $A_1$ , $A_0$. If the operation is a write operation on a BIU register, B/O*, $A_1$ and $A_0$ are decoded to check the word size, the NBC enables WSOE (Word Size Out Enable) to pass the control information to the EDAC. After generating the check bits, the EDAC passes the control information to the NuFTbus.

The acknowledgement coming in, is latched by the NBC using NLWSI. Then it is passed to the EDAC by enabling NWSOE, for checking its validity. The acknowledgement consists of $TM_1$ and $TM_0$. These are decoded by a 2 to 4 decoder to determine the type of acknowledgement. It can be TRFCOM (transfer complete), ERR (error), TMOUT (time out) or TRYLAT (try later).

When the BIU is in the slave mode, the control information is latched by the NBC using the NLWSI signal. The word size (byte, half word or word), is determined using a 3 to 8 decoder.

At the end of the transaction, an acknowledgement is generated on the $GTM_1$ and $GTM_0$ lines. It is stored by the LA (Latch Acknowledgement) signal. It is passed through the EDAC and then on the NuFTbus, the using AOE (Acknowledgement Out Enable) and EDACOE2 signals, respectively.

## 7.3 Implementation of BTC and NBC controllers

The BTC and the NBC control units are converted into a form acceptable by the HIGHLAND system by using some of the Berkeley CAD tools. The tools used are:

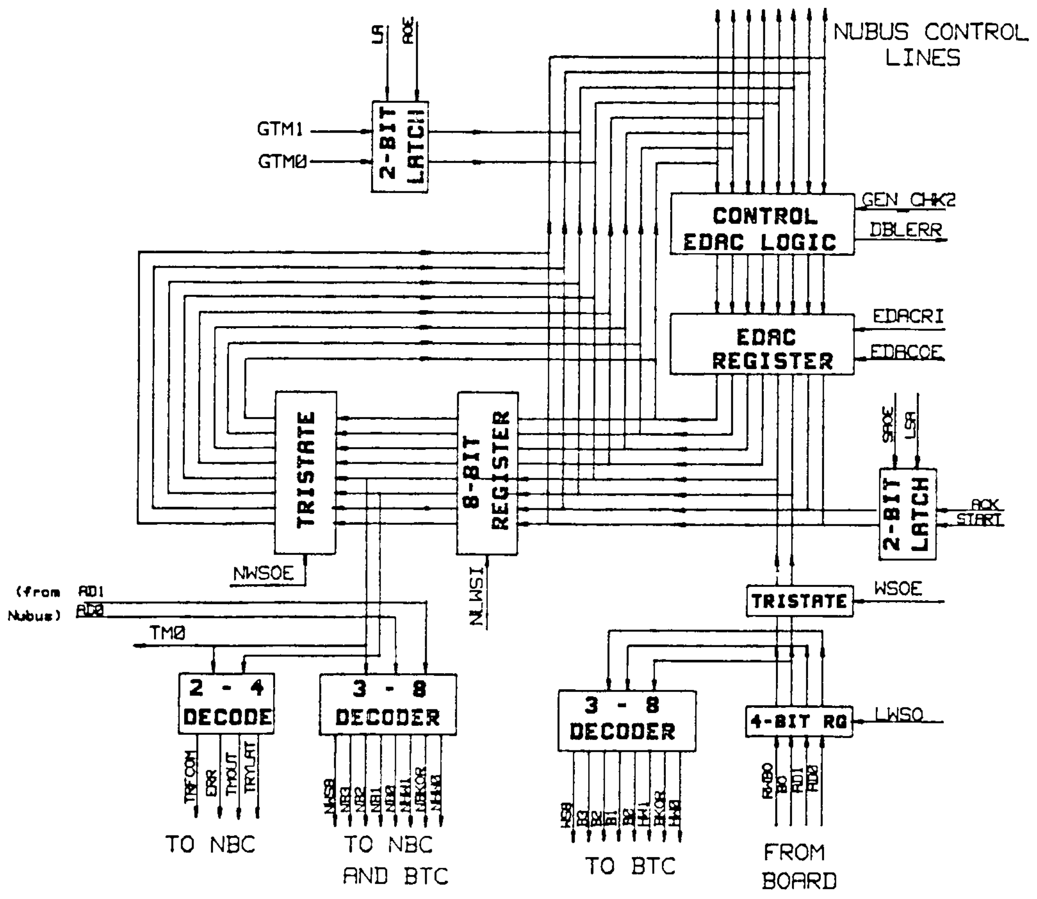


Figure 64. Control path

1. PEG (PLA Equation Generator): PEG is a finite state machine compiler. It takes the description of a state machine as input and converts it into boolean equations which implement the state machine.

2. EQNTOTT (Equation to Truth table): EQNTOTT generates a truth table suitable for PLA programming from a set of boolean equations. The output of PEG can be directly fed to EQNTOTT.

3. ESPRESSO: ESPRESSO is a minimization program which takes a 2-level representation of a boolean function as input and produces a minimum equivalent representation. ESPRESSO accepts output of EQNTOTT and reduces the minterms in the truth table generated for PLA implementation.

   Signal names are not listed by ESPRESSO. After the signal names are added to the ESPRESSO output and the the state vector bits specified, the output of ESPRESSO can be accepted as an input by a program LOGEN on the HIGHLAND system. LOGEN produces a corresponding code in terms of McLDL.

### 7.3.1 PEG

PEG implements the state machine as a Moore machine. In a Moore machine, the outputs are dependent only on the present state of the machine and not on the inputs. Thus, the outputs of the state machine change after the clock pulse.

The syntax for PEG is quite simple.The input consists of a sequence of states. Each state is represented by a colon. The signals which are supposed to go high in a state are listed in an ASSERT statement in that state. The control is transferred to the next sequential state, unless the transfer is explicitly defined by an IF_THEN_ELSE, CASE or a GOTO statement. The state is changed with each successive clock pulse. Figure 65 shows a PEG input file. PEG generates the combinational logic which implements the state machine. It generates n state bits, (*Inst*0$^x$-*Instn*$^x$ and *Outst*0$^x$ -*Outstn*$^x$) corresponding to n flip flops needed to implement the state machine. The flip flops have to be placed externally with the necessary clock input to complete the implementation of the state machine, (Figure 66). Outputs *Outst*0$^x$-*Outstn*$^x$ are connected to the D inputs of the flip flops and inputs *Inst*0$^x$-*Instn*$^x$ are taken from the Q outputs of the flip flops.

### 7.3.2 Difficulties encountered while using PEG

#### *7.3.2.1 Multiple clocks*

As explained in the previous section, flip flops have to be placed externally to combinational logic generated by PEG to complete a state machine representation. N flip flops define up to $2^N$ states. A common clock is given to all the flip flops. BTC uses a single 40 MHz internal clock for all its operations, but different sequences in NBC use different clocks. All signal assertions on the NuFTbus are done on the rising edge of BUSCLK (10 MHz) while the NuFTbus signals are sampled on the falling edge of BUSCLK. All the other operations use a 40 MHz. internal clock to speed up the transaction. The use of different clocks for different sequences in the program leads to difficulties while implementing the circuit in PEG. The state bits are decoded in the combinational logic generated by PEG. Thus, within the same program, the states in the different sequences (using different clock) can be identified, but there is no provision for assigning different clocks for different sequences of states. Thus, to make possible the use of three different clocks, the NBC state machine is broken down into three state machines. NBCRI uses

```
-- simple D flipflop

INPUTS   :  RESET d clk clear;
OUTPUTS  :  q qb;

start    :  IF NOT clear THEN LOOP;
wait     :  CASE (clk d)
                 0 ? = >dummy;
                 1 0 = >ns1;
                 1 1 = >ns2;
              ENDCASE = >start;

dummy    :  GOTO wait;
ns1      :  ASSERT q; GOTO wait;
ns2      :  ASSERT qb; GOTO wait;
```

Figure 65. A sample PEG input

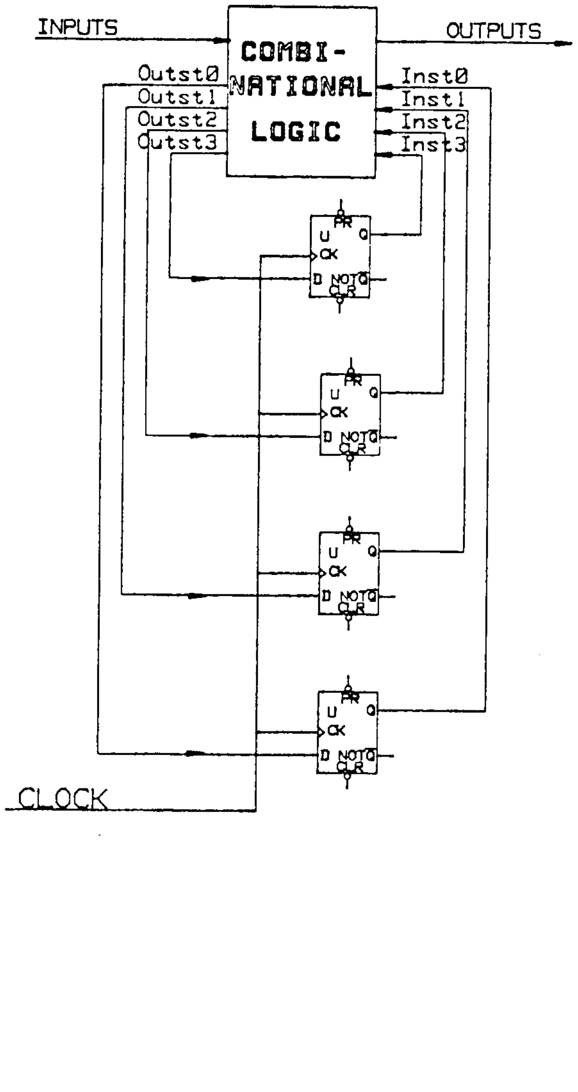


Figure 66. Forming state machine by connecting flip flops

the rising edge of internal clock while NBCRB and NBCFB use the rising and falling edges of BUSCLK, respectively.

Transitions from one state machine to another are frequent. Consider the following example to understand the transitions between the machines. Let the first machine be in a state X. Let the control be transferred to a state Y in the second state machine. Once the operation in the second machine is over, the control may not come back to the state X or the state after X in the first machine, but it can go to some other state of the first machine or to the third machine also. So the first machine cannot wait in the state X. Thus, a way to transfer control to different entry points in different machines is needed. This is achieved by introducing dummy signals.

Consider the case in which state machine NBCRI is in a state D12 and the control is to be transferred from NBCRI to a state D5 in state machine NBCRB. When NBCRI is in state D12, it sets a dummy signal DUM5 as an output. Then it goes into the idle state, ZERO, and monitors the number of dummy input signals (using a CASE statement). The dummy input signals are the entry points to different states in NBCRI. DUM5 is a dummy input to NBCRB. Once DUM5 is set, with the next clock pulse NBCRB goes from its idle state to the state D5 and starts executing the sequence of operation. Once the sequence is finished, NBCRB sets another dummy variable (an entry point for NBCRI or NBCFB), and goes to its idle state.

Thus, even if the control is transferred from one machine to another, no clock pulses are missed during the transition. This method adds more input and output signals to each machine but reduces the number of states and hence reduces the delay, if compared to the method of decoding at every entry point.

#### 7.3.2.2 *Set-reset for output signals*

PEG does not have a global ASSERT statement. A signal listed in the ASSERT statement goes to level 1 and remains high only while the machine is in that state. Then it goes back to level 0. So, if a signal is to be high for four consecutive states, it has to be ASSERTed in each state. This leads to a problem when internal flags are set, or when there is a state transition across two state machines. A state which is an entry point can be entered from different states and the signals in the ASSERT list of those states may not be the same. To overcome this problem an external set-reset flip flop is added for each of these signals. When the output is supposed to go high, an ASSERT statement in that state sets a corresponding flip flop. Then that signal is not ASSERTed in the subsequent states. When that signal is supposed to go low after a state, the flip flop is reset by a signal ASSERTed in that state. The internal flags used to distinguish between single or block operations are also implemented in the same way.

#### 7.3.2.3 *Bidirectional signals*

PEG does not allow bidirectional signals while describing a machine. Both the NBC and the BTC use flags (as explained before) for various operations. These flags are set at the beginning of the transaction and are tested at different points during execution. Thus, flags have to be bidirectional signals. An external set-reset flip flop is provided for each bidirectional signal. Let a flag be FLAG1. It is set by a signal FLAG1 and reset by NFLAG1. A corresponding input IFLAG1 is provided, which is connected to FLAG1, external to the machine. This makes FLAG1 a bidirectional signal.

The code written as an input to PEG, for all four state machines is presented in the reference [20].

# 8.0 INCORPORATING TESTABILITY

## 8.1 Introduction

Due to the ever increasing complexity of VLSI design, different types of circuits require different approaches to test the functionality of the circuits. To ease the process of testing, the circuits are designed incorporating some testability features. Several different techniques of designing for testability are proposed and widely used [25]. These techniques can be broadly divided into two categories [26]. The first category is ad hoc techniques which give guidelines for making a particular circuit testable. These techniques cannot be generalized to fit different types of design. Ad hoc techniques include partitioning a design into different functional blocks for easier testing and the addition of test points. The second category consists of techniques with more structured or generalized approaches which provide a set of rules for incorporating testability in the design. These techniques include scan path, LSSD and signature analysis.

For incorporating testability in this design, a combination of an ad hoc approach and a structured technique, such as scan path, is considered. The basic goals for adding testability features to a circuit are [27]:

1. the circuit can be put into a desired initial state
2. the internal states of the circuit can be easily controlled through the application of test inputs

3. the internal states of the circuit can be easily determined by observing the outputs of the circuit and by using special test points.

Before explaining the testability features incorporated in the design, some of the techniques are briefly described below.

## 8.2 Test points

Test points are the internal nodes of the circuit which are brought out on the pins of an IC chip or through accessible locations on a PC board in order to observe the circuit's internal states. Test points can function as both the inputs as well as outputs. Test points enhance controllability as well as observability of the circuit. As an input, a test point can be used to apply a test pattern and to control the operation of the circuit. If the test point is used as an output, it enables the state of the internal node to be seen at the chip output. The ability to apply test patterns to a circuit via primary inputs to control the the operation of the circuit is referred to as controllability. The ability to observe the internal nodes of the circuit at the IC chip output is referred to as observability [28].

## 8.3 Scan path technique

The scan path technique is used to test a sequential machine. A scan path circuit has two modes of operation, a normal mode and a scan or test mode. In the test mode, the flip flops of the sequential machine are connected together to form a shift register. This enables the shifting of a test pattern into the state machine. For a state machine implemented with n flip flops, the machine can be put into a desired state by shifting in n bits through a scan-in line. Also, by shifting out n bits from the output of the last flip flop in the scan chain and observing them at the scan-out output, the present state of the machine can be determined. The sequence of operation for testing a given circuit is as follows:

1. put the circuit in the shift register scan mode
2. check the operation of the shift register by using the scan-in, scan-out and scan clock signals (input a test pattern to test the registers and be sure that the pattern is seen at the output)
3. feed in the initial or desired state of the machine through the scan-in input
4. put the circuit in the normal mode of operation
5. apply an input pattern to the circuit (the circuit will change the state depending up on the inputs)
6. put the circuit back in the shift register mode and shift out the new state of the machine
7. repeat this procedure to test all the states

The following section describes the testability techniques used in the design of this BIU.

## 8.4 Testability features of the BIU

### 8.4.1 Test points

The two controllers, the NBC and the BTC, carry out all of the operations of the BIU by selecting a proper sequence of actions for each operation. While executing these operations, the BTC and the NBC interact with each other. So, signals on the NBC-BTC interface are defined to be test points. These signals are:

- TIP
- AIR
- DIR
- TOR
- RFD
- BBLOCK
- BNACK
- BNLOCK
- BBACK

The states of the NBC and the BTC can be determined by observing these test points. Tri state buffers are provided internally on these lines, (Figure 67). The buffers are enabled during normal operation. The buffers are also enabled when the test points are used as outputs. The buffers are disabled when these test points are used as an input. Once the buffers are disabled, the lines (Figure 67). become tristate and these signals can be set or reset externally. In this way, individual parts of the circuit (either BTC or NBC) can be tested independently.

Other important internal signals defined to be test points are ARBERR, COMP, and internal GRANT signals from the arbitration logic. Also, the enable/disable signals for the board and the NuFTbus transceiver are brought out. The transceivers can be enabled, whenever necessary, to check the addr/data word present on the internal bus. The

BIU status register is made accessible externally so that the different flags set by the BTC or the NBC can be checked.

### 8.4.2 Scan path for controllers

For both the NBC and the BTC, separate scan-in and scan-out signals are provided. As compared with connecting all the flip flops in one big shift register, using two or more shift registers requires less time to shift in or shift out the states of the machine. The connection of the NBC and the BTC registers in the form of shift register is shown in Figure 68. The NBC and the BTC do not use the same clock for all operations. Therefore, a separate scan clock is used. (The internal clock, which is the clock for the BTC, can also be used as a scan clock for the NBC).

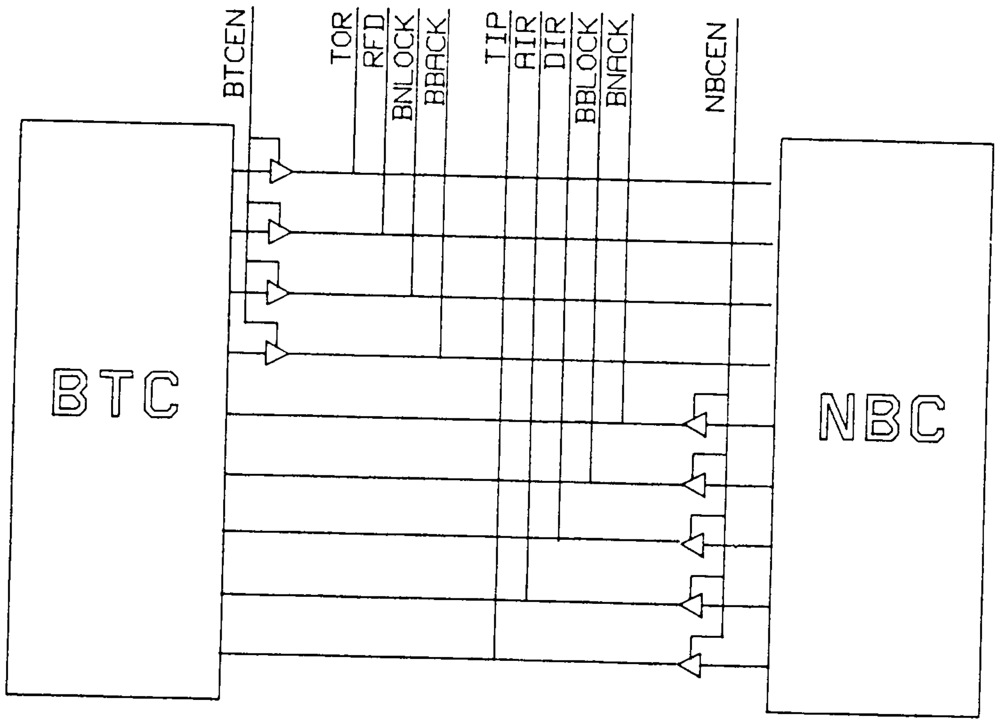


Figure 67. Test points

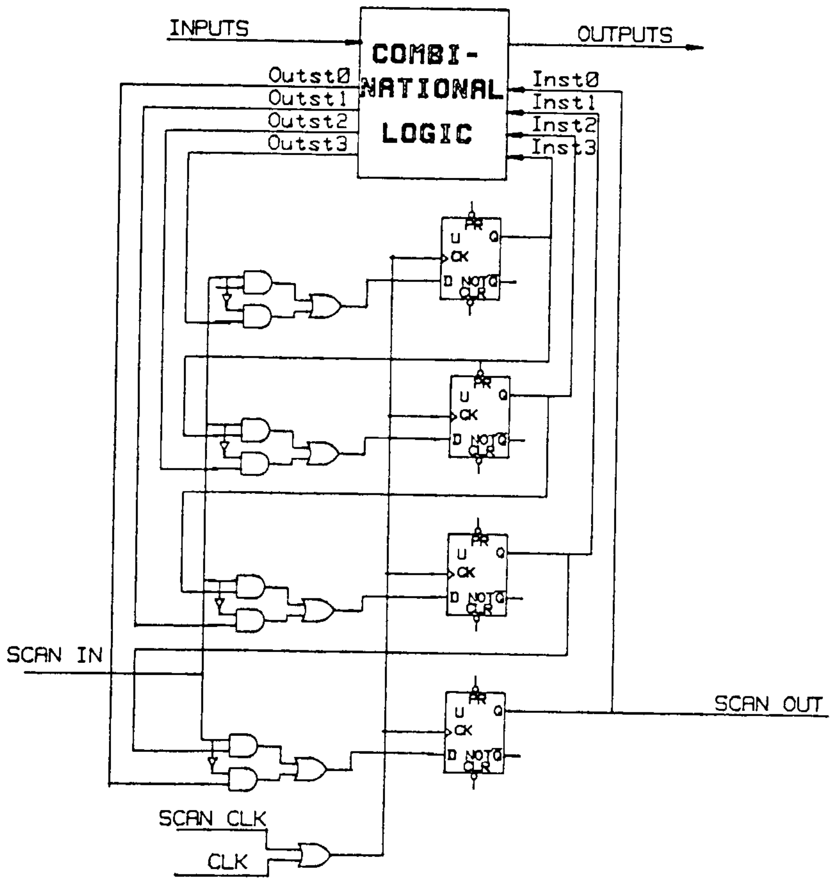


Figure 68. Scan path for state machine

# 9.0 CONCLUSION

The purpose of this thesis was to develop a bus interface unit for a parallel system bus in a multiprocessing environment. 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.

Several different 32-bit parallel bus architectures were evaluated and the Nubus architecture was chosen for this application. Major factors in favor of using the Nubus were that it has a simple protocol, that it has the fewest number of signal lines of all the buses those were evaluated, and that it uses a fair arbitration mechanism. To incorporate upset tolerance properties, additions were made in the basic Nubus signals. In a multiprocessing environment, arbitration among different BIUs for bus mastership is one of the important criteria in the proper functioning of the system. Hence, the arbitration unit was duplicated in each BIU to provide for redundant fault tolerance. It was made mandatory for the BIU to win the arbitration (get a local grant signal) in both redundant modules before a bus transaction could proceed.

Simulation results for the arbitration process proved that any arbitration contest even in the presence of a single bit error, always ended up with only one module becoming the bus master. Also, the bus was never left without a master.

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.

The design of the BIU control unit(s) was done at a register transfer level using the hardware description language VHDL, as a vehicle for describing the design. The control units were simulated extensively to verify the performance of the various transactions on the Nubus. Efforts were made to meet all the specifications defined by the Nubus standard. The functionality of the BIU was tested using the VHDL simulator. Different NuFTbus transactions such as single word transfers and block transfers were simulated and verified. Simulation results obtained for different bus transactions matched the data transfer specifications outlined in the Nubus standard. The simulations results for the arbitration unit and the BIU control units verified the functionality of the design.

## 9.1 Future work

The operation of the BIU as a whole needs to be tested before actually fabricating the design as a VLSI chip. Chapter 8 presented a way in which the design could be implemented as a gate array. The complexity of the design was determined from the preliminary implementation work. The gate count needed to implement the state machines is about 10,000 gates and the gate count for the remaining circuitry is about 11,000 gates. The HIGHLAND system can support a design with 25,000 transistor pairs (about 11,000 gates) in one gate array. The design could be broken up into two gate arrays such that the controllers would go in one gate array and the other circuitry along with the data path could be in the second gate array. The interface between the two gate arrays would consist of the control signals from the controllers in one gate array to the rest of the circuit in the other gate array.

Two factors should be considered while evaluating the performance of the BIU. First, the functionality should be checked with respect to different transactions in an error free environment. Secondly, the upset tolerance needs to be checked by monitoring the response of the BIU to single bit errors.

A test bench for the BIU could consist of an Intel 80386-based board with which the BIU could communicate. The interface of the microprocessor board with the BIU could be tested using the testability features incorporated into the BIU. Two such boards could interact with each other to verify the overall operation. To test the upset tolerance of the BIU, a similar procedure could be repeated while transient faults are introduced. The transient upsets could be introduced by placing the board in a radiation chamber, or more practically, by introducing power supply glitches on signal lines, which would create a similar effect.

# BIBLIOGRAPHY

1. W. B. Leonard, K. K. Chow, "A new bus architecture for distributed avionic systems", *IEEE AIAA 5th Digital Avionics Systems Conference*, 1983, pp 12.5/1-6.

2. D. B. Gustavson, "Computer buses - A tutorial", *IEEE Micro*, August 1984.

3. P. L. Borrill, "Microstandards special feature : A comparison of 32- bit buses", *IEEE Micro*, December 1985.

4. J. C. Pickel, J. T. Blandford "Cosmic ray induced errors in MOS memory cells", *IEEE transactions on nuclear science*, Vol NS-25, No 6, December 1978, pp. 1166-1170.

5. J. P. Woods, D. K. Nichols, W. E. Price, "Investigation for single event upsets in MSI devices", *IEEE transactions on nuclear science*, Vol NS-28, No 6, December 1978, pp. 4022-4025.

6. B. Nicholson, "Synchronous 32-bit backplane buses open up distributed system design", *EDN*, June 14, 1984.

7. Macintosh II and Macintosh SE cards and drivers, APDA Draft, March 2, 1987, Apple Technical Publication.

8. S. Ohr, "Three 32-bit-wide buses will give 32-bit uCs main frame performance", *Electronic Design*, January 12, 1984.

9. W. Fischer, "IEEE P1014 - A standard for the high performance VME bus", *IEEE Micro*, February 1985.

10. The VMEbus Specifications Manual, revision C.1, October 1985, Motorola Microsystems.

11. Multibus II bus architecture specification handbook, Intel Corporation.

12. Nubus - a simple 32-bit backplane bus, P1196 specification, Draft 2.0, P1196 working group of microprocessor standards committee, IEEE, December 15, 1986.

13. Nubus specifications, Data systems group, Texas Instruments, Inc., Part No 2242825-0001.

14. G. P. White, "Battle of the buses for 32-bit systems", *Systems and Software*, September 1984.

15. J. Theus, "Asynchronous operation boosts longevity", *EDN*, February 1984, pp. 72-73.

16. NRL meetings with G. Flach, R. Higgins, J. Golba, Washington, D.C., June, August, September 1987.

17. M. Ercegovac, T. Lang, "Digital Systems and Hardware/Firmware Algorithms", John Wiley and Sons, 1985.

18. The TTL Data Book, Volume 3, Texas Instruments, 1984.

19. J. G. Tront, P. G. Paranjape, "BIU for NuFTbus", Interim project report, NRL Project Research Group, Virginia Tech, Blacksburg, Virginia, June 1987.

20. J. G. Tront, P. G. Paranjape, "Controllers for NuFTbus", Interim project report, NRL Project Research Group, Virginia Tech, Blacksburg, Virginia, January, 1988.

21. J. R. Armstrong, "Chip level modeling with VHDL", Prentice/Hall International, 1988.

22. PC-LOGS User's Manual, Personal CAD Systems, Inc.

23. HIGHLAND Reference Manual, United Technologies Microelectronic Center, publication 94002.

24. 1986 VLSI Tools : Still more work by original artists, Report # UCB/CSD 86/272, Computer Science Division, University of California, Berkeley, December, 1985.

25. T. W. Williams, K. P. Parker, "Design for testability survey", *IEEE transactions on Computers*, Vol. C-31, No.1, January 1982.

26. T. W. Williams, K. P. Parker, "Testing logic networks and design for testability",*Computer*, October 1979, pp. 9-21.

27. S. Funatsu, N. Wataksuki, A. Yamada, "Designing digital circuits with easily testable considerations", *Conf. records, 1978 Semiconductor Test Conf.*, pp. 98-102.

28. P. K. Lala, "Fault tolerant and fault testable hardware design", Prentice/Hall International, 1985.

# Appendix A. Detailed BIU block diagram

This block diagram is attached to the back cover of this thesis.