1-1-1990

The Application Of RISC Processors To Training Simulators

Thomas L. Clarke

INSTITUTE FOR SIMULATION AND TRAINING

FHTIC Final Project Report

1-1-90 to 12-31-90

THE APPLICATION OF RISC PROCESSORS TO TRAINING SIMULATORS

Author:
Thomas L. Clarke

Institute for Simulation and Training
12424 Research Parkway, Suite 300
Orlando FL 32826

University of Central Florida
Division of Sponsored Research

IST-XX-89-#

State of Florida
HIGH TECHNOLOGY AND INDUSTRY COUNCIL
APPLIED RESEARCH GRANTS PROGRAM

RESEARCH REPORT

FHTIC Contract No. ______________ Date Submitted: ______________

Report Title: The Application of RISC Processors to Training Simulators

Report Author(s) - name and address of organization:
Thomas L. Clarke
IST/UCF
12424 Research Parkway, Suite 300
Orlando, FL 32826

Type of Report: ____ Quarterly Project X Final Project
____ Semiannual Fiscal ____ Final Fiscal

Period Covered: From 1-1-90 to 12-31-90 ; Number of pages 256

ABSTRACT of contents of this submission (150-200 words; for Project Reports only)

Despite very successful research, this project was not selected for renewal beyond 1990 because of the intense competition for FHTIC funds. Nevertheless, a number of key technologies have been developed to the point at which they can attract funding from other agencies to support advanced development. In particular, a detailed hardware design for interfacing transputer hardware to the NeXT computer was completed under this project and may form the basis of a commercial product.

A study on the utility of RISC processors as the control computer in a training simulators was also completed. Results of this study should be of value to Florida simulator manufacturers. Both the research into transputers and RISC processors provided the basis for a proposal to DARPA in the area of advanced visual displays.

The best results from this project will not be abandoned with the end of FHTIC funding, but will be supported by other means or will utilized in the companion Terrain Data Bases for Simulation Project.

Subject Key Words (list 3-5): RISC, transputer, parallel processing

For FHTIC Use Only

Date Received: ______________ **FHTIC Publication Code:**

Summary

This project was not selected for renewal beyond 1990. It has nevertheless been a very successful project. Most importantly, a number of key technologies have been developed to the point at which they can attract funding from other agencies to support advanced development.

A detailed hardware design for interfacing transputer hardware to the NeXT computer was completed as a master's thesis under this project. This thesis is included as Appendix II.

A study of the utility of RISC processors as the control computer in a training simulators was completed. The results of this study should be of value to Florida simulator manufacturers. A copy of the paper detailing these results is included as Appendix IV.

The research into transputers and RISC processors provided the basis for a proposal to DARPA in the area of advanced visual displays. These displays will be particularly useful in the area of virtual reality which is one of the fastest growing application areas.

The best ideas from this project will not be abandoned with the end of FHTIC funding, but will be utilized in the Terrain Data Bases for Simulation Project. In addition other sources of funding continue to be aggressively pursued.

Significant Research Findings

A detailed hardware design for interfacing transputer hardware to the NeXT computer was completed as a master's thesis under this project. This thesis is included as Appendix II. This design will bring the expandable processing power of arrays of transputers to the growing body of NeXT machine uses. No industrial partner has, however, yet been identified to produce this design.

Work was completed on the simulator benchmark supplied by Naval Training Systems Center (NTSC). The C-code derived from the original FORTRAN was polished and prepared for extensive benchmark runs.

The TCP/IP workstation network at IST includes RISC machines using a variety of processors. The Sun uses SPARC, the Silicon Graphics uses MIPS 2000, and Data General uses Motorola 88000. There are in addition machines on the network using CISC processors such as 80386 and 68030. In addition IBM made one of their new RS/6000 RISC machines available for benchmarking.

Extensive comparison benchmarks were made available for presentation at the 1990 Interservice/Industry Training Systems Conference. The results of this study are included as Appendix III.

A major result of this project has been a detailed literature survey to identify previous work related to applying parallel processing technology such as transputers to the problem of generating imagery. In general there are two broad approaches to generating images. Broadly speaking, these are ray tracing and polygonal rendering. A brief discussion of some of the most valuable references follows.

Andrew Glassner of Xerox PARC (Palo Alto Research Center) has edited a book An Introduction to Ray Tracing that provides a broad survey of the field. Of particular interest is the stochastic sampling approach of Cook ("Stochastic Sampling and Distributed Ray Tracing"). It avoids aliasing artifacts by redistributing under-sampled energy across the spectrum. Stochastic casting or rays is a good way to divide effort among an array of transputers.

In the same volume, Arvo and Kirk ("A Survey of Ray Tracing Acceleration Techniques") discuss various methods of speeding ray-tracing including the use of vector and parallel architectures. A rather complete ray-tracing bibliography is presented in the article by Heckbert and Haines ("A Ray Tracing Bibliography").

Another valuable collection is Parallel Processing for Computer Vision and Display edited by Dew, Earnshaw and Heywood (University of Leeds and IBM, Great Britain). Their book contains a variety of algorithms oriented toward speeding visualization using parallel computation. In particular the article by Caspary and Scherson ("A self-balanced parallel ray-tracing algorithm") presents a parallel ray-tracing algorithm.

The article by Holliman, Morris, Dew, and de Penington ("An evaluation of the processor farm model for visualizing constructive solid geometry") discusses the use of a "processor farm" for visualizing solid geometry. This article is particularly interesting since the processor farm is implemented using transputers.

Another book that has proven a valuable source of material is Image Synthesis by Magnenat-Thalmann and Daniel Thalmann (University of Montreal). They present a variety of techniques for dealing with texture that are applicable to the problem of generating images using arrays of transputers.

Dew, P. M., R. A. Earnshaw, and T. R. Heywood (editors), 1989; Parallel Processing for Computer Vision and Display; Addison-Wesley Publishing Company, New York, 503pp.

Glassner, Andrew S. (editor), 1989; An Introduction to Ray Tracing; Academic Press, New York, 327pp.

Magnenat-Thalmann, N. and D. Thalmann; 1987; Image Synthesis; Springer-Verlag, New York, 400pp.

The literature search uncovered recent research of some importance. Two researcher's at NYU's Courant Institute have identified the usefulness of the mathematical technique of conformal mapping for performing the transformations needed in a CIG.

Frederick, Carl and Schwartz, Eric. L. 1990. "Conformal Image Warping", IEEE Computer Graphics and Applications, March , 54-61.

Earlier work on conformal mapping (for non-visual applications) shows promise for adapting this transform to parallel hardware such as the transputer:

Trefethen, Lloyd N. 1980. "Numerical Computation of the Schwarz-Christoffel Transformation", SIAM J. SCI. STAT. COMPUT., 1, 82-102.

Adoption of conformal mapping to transputers will be a significant research contribution that will have applications beyond the visual transformation for a CIG.

For example, a hemispherical display (2π steradians) achieving the 1 minute of arc resolution of the human eye would require 75 million display elements. This is beyond any foreseeable, affordable technology. In addition, updating the 75 million pixels fast enough to avoid flicker and other artifacts would require >10,000 MIPS.

If the warping algorithm is implemented in a local, pixel by pixel, fashion , then good use can be made of parallel hardware such as arrays of transputers. This insight was a key factor in the proposal to DARPA (Appendix IV).

Comparison with Goals

This project was not selected for renewal beyond 1990. It has nevertheless been a very successful project. Most importantly, a number of key technologies have been developed to the point at which they can attract funding from other agencies to support advanced development.

A detailed hardware design for interfacing transputer hardware to the NeXT computer was completed as a master's thesis under this project. This thesis is included as Appendix II.

A study on the utility of RISC processors as the control computer in a training simulators was completed. The results of this study should be of value to Florida simulator manufacturers. A copy of the paper detailing these results is included as Appendix IV.

The research into transputers and RISC processors provide support for a proposal to DARPA in the area of advanced visual displays. This displays will be particularly important in the area of virtual reality which is one of the fastest growing application areas.

Commercialization Activities

At the close of the year, the RFP for DOD SBIR research contained several topics that would provide a good vehicle for collaboration with local small businesses. Discussions were held with Daedalian, Inc (Russ Hauck) concerning commercialization of the technologies resulting from this research. Joint proposals to NTSC and the AF were being prepared.

Current Problems

There are no major problems with this project, nor are any anticipated.

Upcoming Milestones

This project has not been selected for renewed funding. Therefore there are no upcoming milestones.

Publications and Presentations

Andres Alverez, a student of Dr. Petrasko, the co-PI, completed a master's degree as a result of this project. A copy of his research presentation is included as Appendix I and a copy of his thesis is included as Appendix II.

A paper , "A Portable Benchmark for Simulator Processors", was submitted to I/ITSC, but did not make the final cut for presentation. The reviewers felt felt that the results of the brickbat benchmark were essentially the same as those of the conventional dhrystone and whetstone benchmarks. The research was too successful!

A copy of this paper is attached as Appendix III.

Direct External Support

DARPA has responded favorably to a draft proposal "Optimal Virtual World Displays" which uses many techniques developed in this research. FHTIC research was fundamental in attracting this interest. The funding level for the DARPA research will be ~$360K over three years. A copy of the proposal is included as Appendix IV.

IBM provided one of their new RS/6000 RISC-based workstations to support benchmarking and image generation activities.

An Aviion RISC workstation loaned to IST by Data General was made available for benchmarking.

Transputer equipment loaned by the Naval Training Systems Center have been used as part of this project. Additional transputers supplied by PM Trade for use in the Visual Systems Laboratory will also be available to this project.

SUS Infrastructure Interactions

This project employed two students at IST, J. Martin Otte, a master's level mathematics student, and Steve Rehfeldt, an undergraduate in EE.

Andres Alverez, a student of Dr. Petrasko, the co-PI, completed a master's degree as a result of this project.

Appendix I

Research Presentation by
Andres Alverez
on
Interfacing Transputers
to the
NeXTbus
in
Partial Fulfillment
of the
Requirements
for the
Degree of
Master of Science

Interfacing an Array of Coprocessors to the NeXTbus

Research Report by:

Andres Alvarez

B.S., University of Florida, 1986

AGENDA

- Introduction to the Coprocessor Architecture
- Introduction to the NeXTbus Architecture
- Introduction to the NeXTbus Interface Chip (NBIC)
- Overview of the Interface Board
- Top–Level Design of Interface Board
- Advantages / Disadvantages of Using the NBIC
- Work Remaining
- Questions and Answers

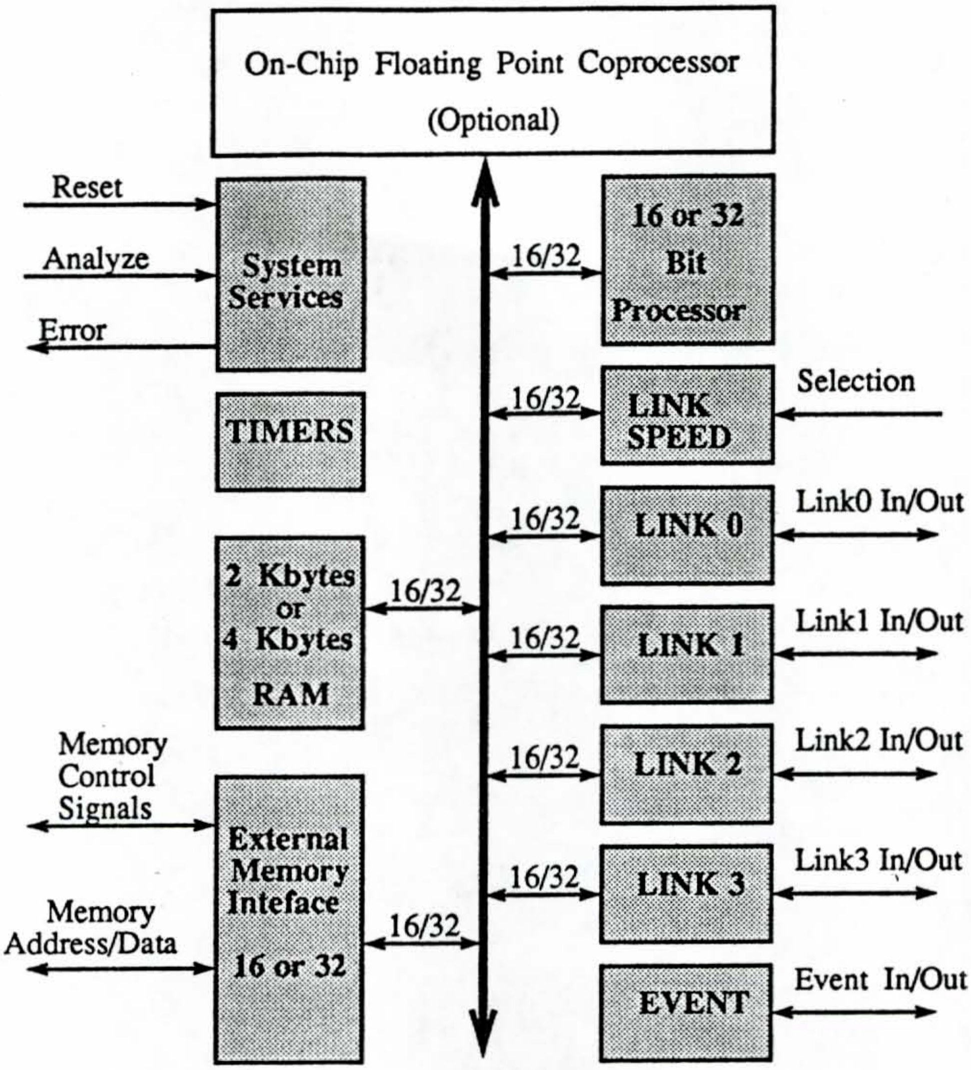
COPROCESSOR ARCHITECTURE

- O Shall be composed of one or several transputer based motherboards.
- O Motherboards are composed of a combination of Inmos transputers, dual–inline transputer modules (TRAMs) and a 32–way link switch.
- O Have the capability to configure different parallel processing networks via link.
- O Powerful array coprocessing networks can be developed by interconnecting several motherboards.

TRANSPUTER INTRODUCTION

- **A high–performance 32–bit microprocessor which contains the following:**
  - ***(1) Its own local memory (2 kbytes / 4 kbytes)***
  - ***(2) 4–high speed serial links***
  - ***(3) A 16 or 32–bit processor***
  - ***(4) Internal timers for real–time processing***
  - ***(5) System services (reset, analyze and error)***
  - ***(6) On–chip floating point unit (optional)***
- **Transputers are hardware processors which execute software processes.**
- **Any number of processes can be executed on a single transputer processor at the same time.**
- **Transputers can be bootstrapped from ROM or via link.**

Block Diagram of Transputer Architecture



Inmos Transputers and their Features

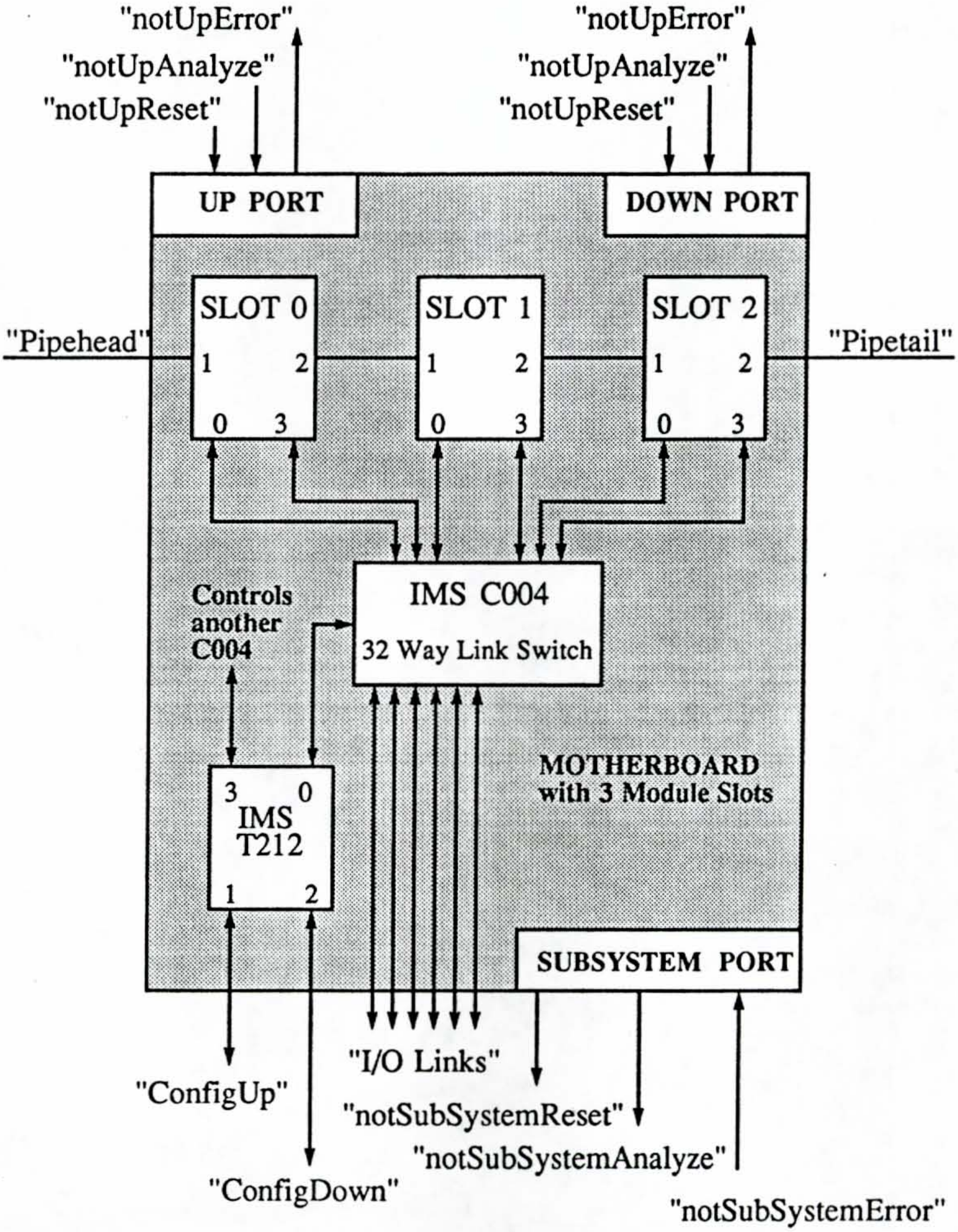
Transputer	*16 or 32*	*ICT*	*On Chip RAM*	*External Memory*	*Resp. to Int.*	*Sustained Data Rate to*	
						External Memory	*Internal Memory*
IMS T805	32	33 ns	4 Kbytes	4 Gbytes	630 ns	40 MB/sec	120 MB/sec
IMS T801	32	33 ns	4 Kbytes	4 Gbytes	630 ns	60 MB/sec	120 MB/sec
IMS T800	32	33 ns	4 Kbytes	4 Gbytes	630 ns	40 MB/sec	120 MB/sec
IMS T425	32	33 ns	4 Kbytes	4 Gbytes	630 ns	40 MB/sec	120 MB/sec
IMS T414	32	50 ns	2 Kbytes	4 Gbytes	950 ns	26 MB/sec	80 MB/sec
IMS T222	16	50 ns	4 Kbytes	64 Kbytes	950 ns	20 MB/sec	40 MB/sec
IMS T225	16	33 ns	4 Kbytes	64 Kbytes	630 ns	30 MB/sec	60 MB/sec

NOTES: Data compiled from Inmos Transputer Data Book. Nomenclature for table.
Resp. to Int. = Response to Interrupts ns = nanosecond
ICT = Internal Cycle Time MB = Mega Bytes

MOTHERBOARD ARCHITECTURE

- Motherboards have a common architecture to allow control and interconnection of several boards.
- Architecture allows different kinds of network to be configured.
- Allow hierarchical control of systems with more than one Motherboard.
- Allow networks to be easily configurable by software.
- Communication can be obtained with the host computer (NeXT) via a single serial link.

Generic Motherboard Top-level Architecture



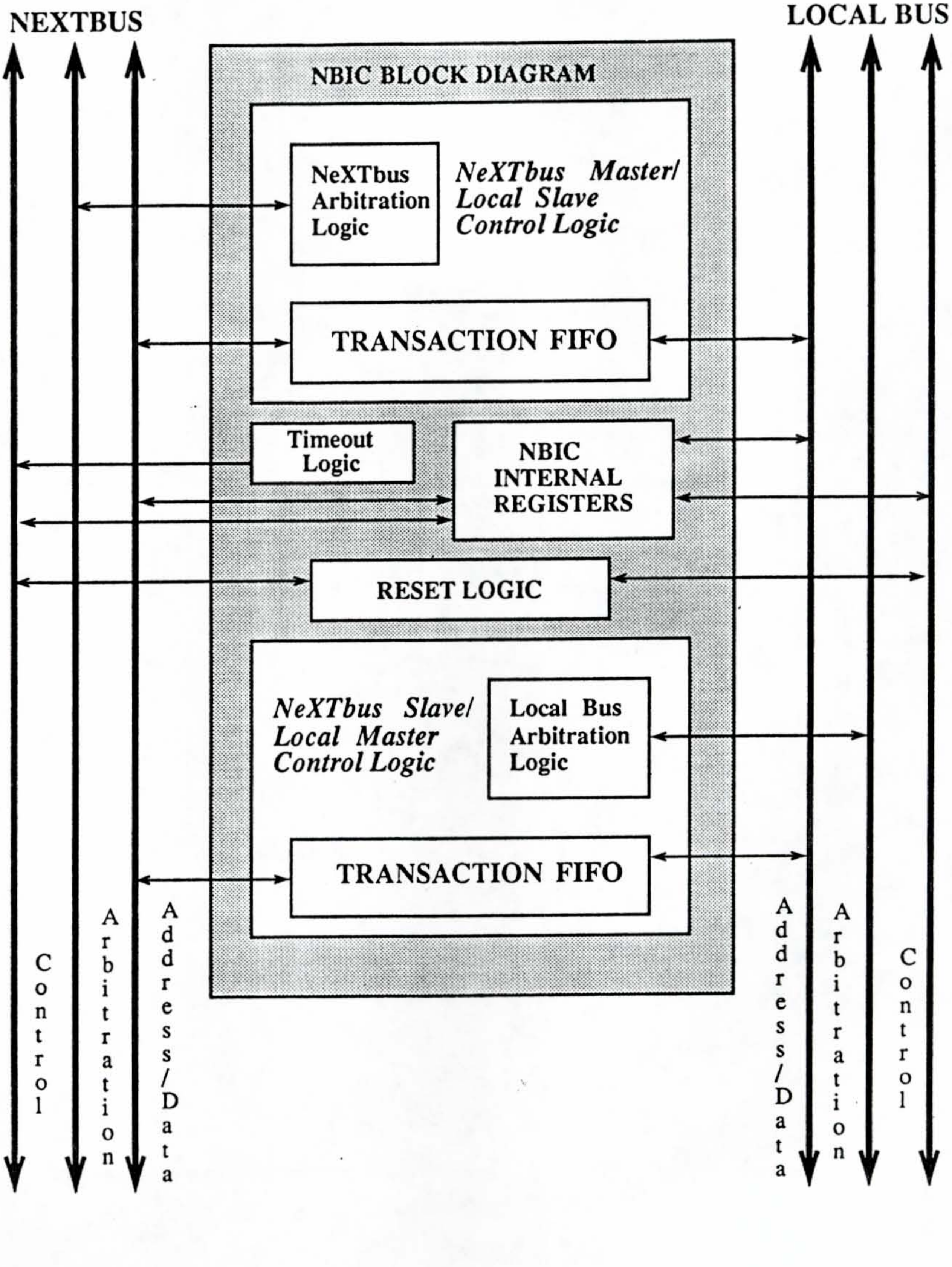
NeXTbus ARCHITECTURE

- A synchronous 12.5 MHz multiplexed bus.
- Bus architecture is composed of 96 signal lines.
- 32–bit addressing (4 Gigabytes of address space).
- Contains Master and Slave Flow Control.
- Has a single–chip interface via the NeXTbus Interface Chip (NBIC).
- Burst transfer size can be 4, 8, 16 or 32 words.
- Each NeXTbus transaction is composed of three basic cycles:
    - *(1) Start Cycle*
    - *(2) Data Transfer*
    - *(3) Acknowledge Cycle*

NeXTbus INTERFACE CHIP

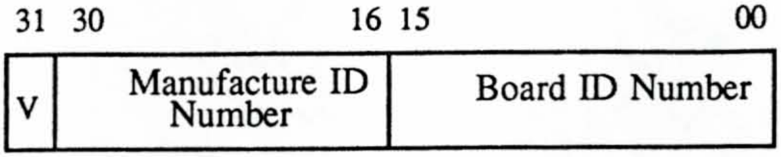
- A 144–pin CMOS VLSI chip.
- Provides a single–chip interface to the NeXTbus.
- Designed to interface specifically with the Motorola 68030.
- Performs Byte–swapping
  - *NeXTbus (Little–Endian Byte Ordering)*
  - *Motorola 68030 (Big–Endian Byte Ordering)*
- Supports both Master and Slave boards.
- Contains five internal programmable registers:
  - *(1) NBIC ID Register*
  - *(2) Control Register*
  - *(3) Configuration Register*
  - *(4) Interrupt Register*
  - *(5) Interrupt Mask Register*
- Provides a local bus interface (used to interface with the array of coprocessors).

Top-level Block Diagram of NBIC

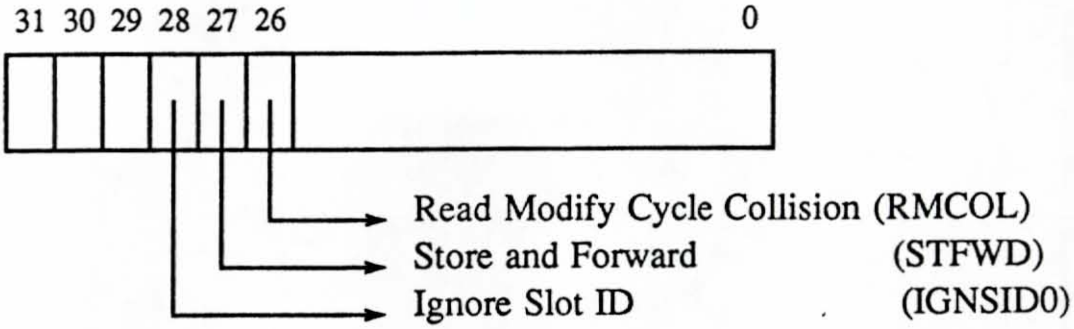


Format for Internal NBIC Registers.

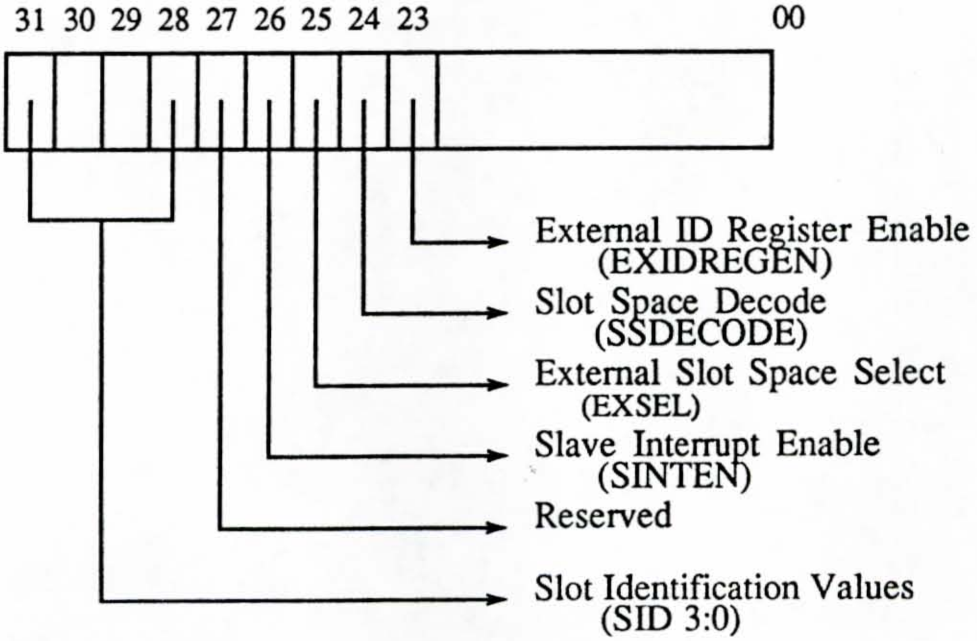
NBIC ID REGISTER



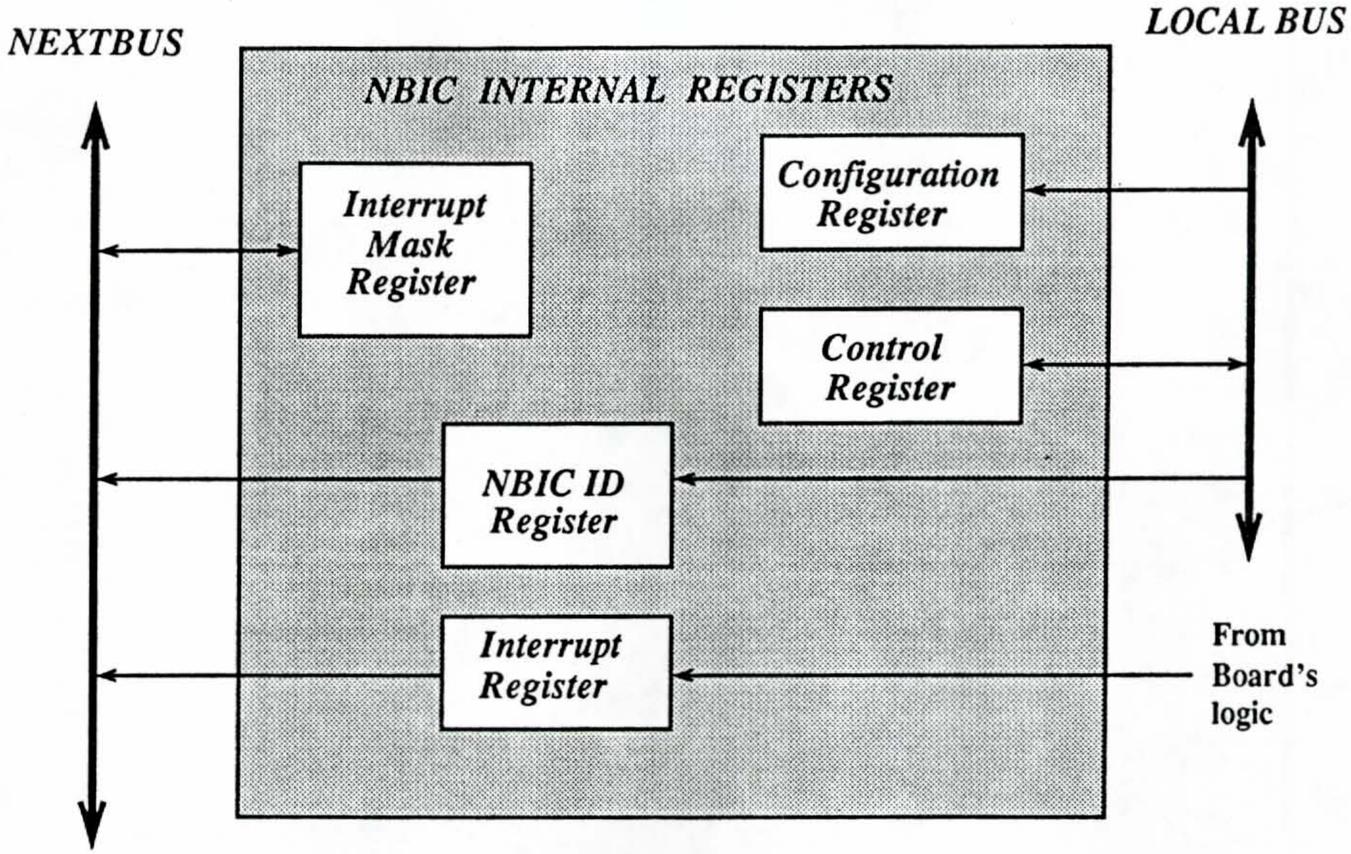
NBIC CONTROL REGISTER


31
30
29
28
27
26
0
Read Modify Cycle Collision (RMCOL)
Store and Forward (STFWD)
Ignore Slot ID (IGNSID0)

NBIC CONFIGURATION REGISTER


31
30
29
28
27
26
25
24
23
00
External ID Register Enable (EXIDREGEN)
Slot Space Decode (SSDECODE)
External Slot Space Select (EXSEL)
Slave Interrupt Enable (SINTEN)
Reserved
Slot Identification Values (SID 3:0)

Access to NBIC Internal Registers.

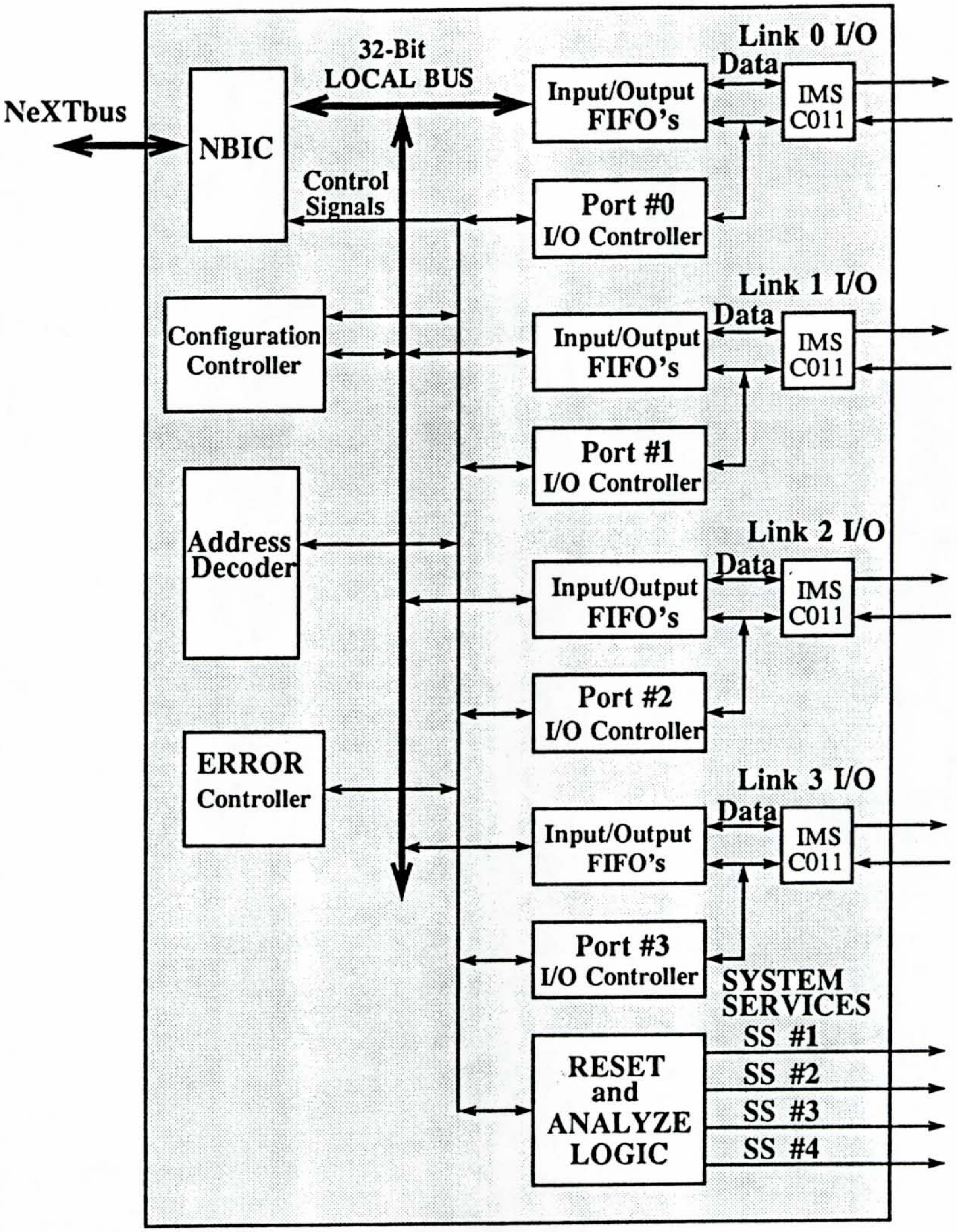


OVERVIEW OF INTERFACE BOARD

Purpose: Allow communication between the NeXT computer and an array of coprocessors via the NBIC.

- O The board shall contain four ports.
- O The ports perform the following functions:
  - *(1) Communicate via the NBIC with the NeXT computer*
  - *(2) Process data which is to be transmitted to the coprocessor network*
  - *(3) Process data which is received from the coprocessor network*
- O The board contains the following H/W modules:
  - *(1) Configuration Controller*
  - *(2) Address Decoder*
  - *(3) Port Controllers (Input and Output)*
  - *(4) Error Controller*
  - *(5) Reset, Analyze and Interrupt Logic*

Top-level Block Diagram of Interface Board



Memory Map of Board

sX4FFFXX sX4000XX	SPECIAL REGISTERS	These Address locations are only valid for Burst Writes.
sX3FFFXX sX3000XX	PORT #3	Burst Byte 3F00 - 3FFF
sX2FFFXX sX2000XX	PORT #2	Burst Byte 2F00 - 2FFF
sX1FFFXX sX1000XX	PORT #1	Burst Byte 1F00 - 1FFF
sX0FFFXX sX0000XX	PORT #0	Burst Byte 0F00 - 0FFF

Mapping of Special Registers.

PORT #0	PORT #1	PORT #2	PORT #3	DESCRIPTION	Type of Register
SUBSYTEM SERVICES					
401D	405D	409D	40DD	Subsystem Analyze	Write Only
4019	4059	4099	40D9	Subsystem Reset/Err	Write/Read
INPUT FIFO (NeXT - to - LINK)					
4015	4055	4095	40D5	FIFO FULL	Read Only
4011	4051	4091	40D1	FIFO EMPTY	Read Only
400D	404D	408D	40CD	FIFO HALF FULL	Read Only
OUTPUT FIFO (LINK - to - NeXT)					
4009	4049	4089	40C9	FIFO FULL	Read Only
4005	4045	4085	40C5	FIFO EMPTY	Read Only
4001	4041	4081	40C1	FIFO HALF FULL	Read Only

CONFIGURATION CONTROLLER

Purpose: Initialize the NBIC ID register and control register at power–up or during a reset sequence.

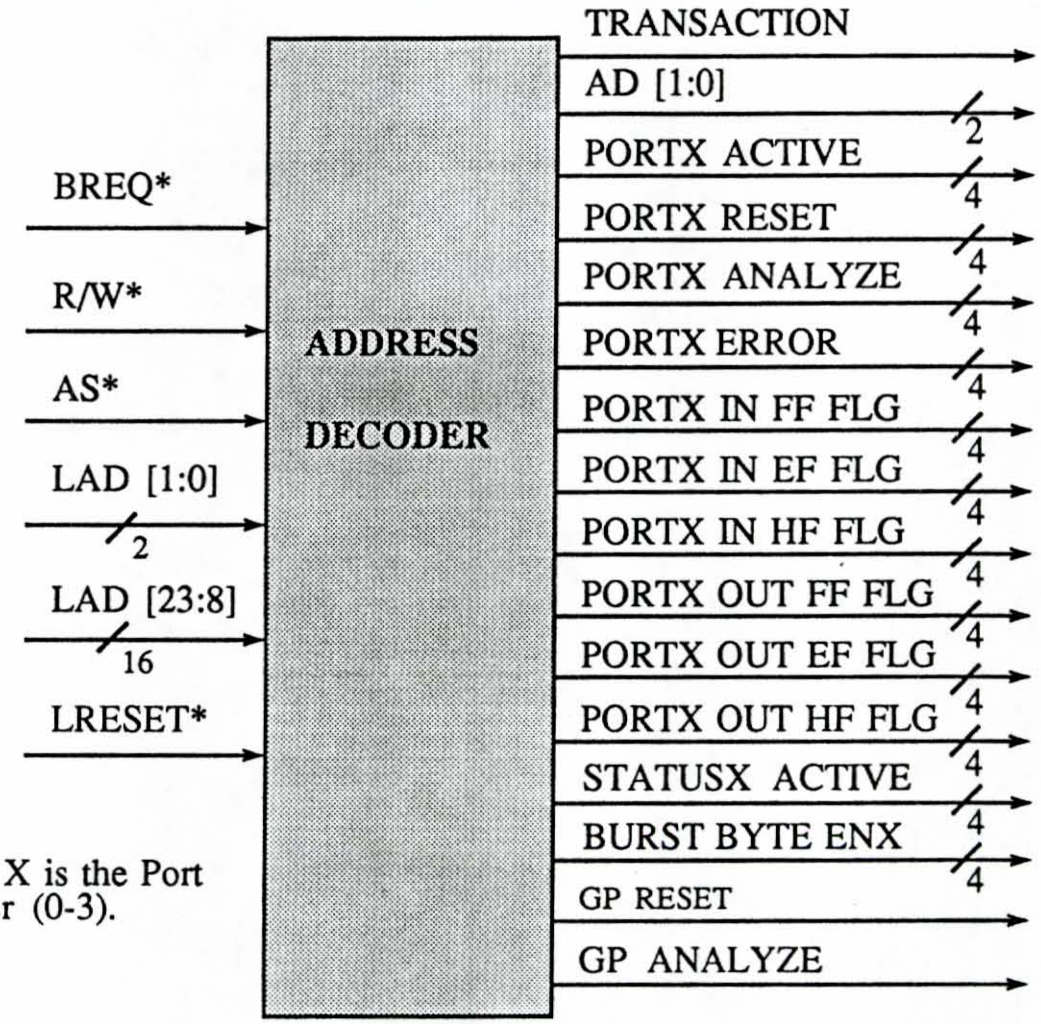
- O NBIC ID Register shall be internal in the NBIC versus external.
- O A PROM shall store the appropriate address and data for the two NBIC registers.
- O Configuration register is configured by board resistors.

ADDRESS DECODER

Purpose: This module shall decode all valid address locations of the board and assert the appropriate signals.

- O This module shall latch the address at the beginning of a transaction.
- O Assert control signals depending on the address lines 2E08–2E23.
- O Address lines 2E23–2E20 select which port is to be active or which special register is to be processed.
- O Address lines 2E00–2E01 used by NeXT to encode type of transaction to be performed.
- O Address lines 2E02–2E06 used by NeXT to encode the burst transfer size during burst transactions.

Top-level Block Diagram of Address Decoder



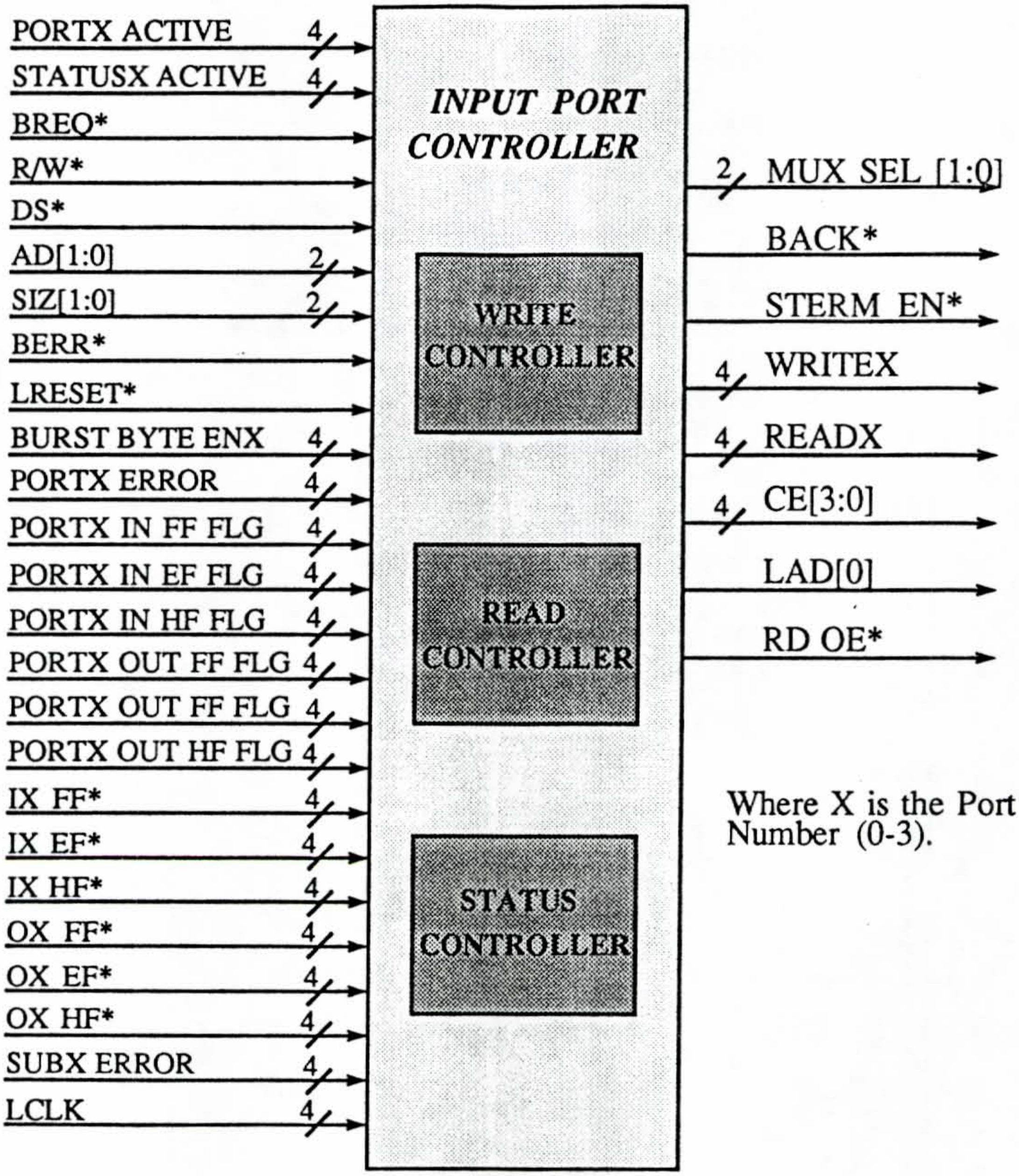
Where X is the Port Number (0-3).

INPUT PORT CONTROLLER

Purpose: Responsible for all transactions between the NBIC local bus and the Input/Output FIFO's.

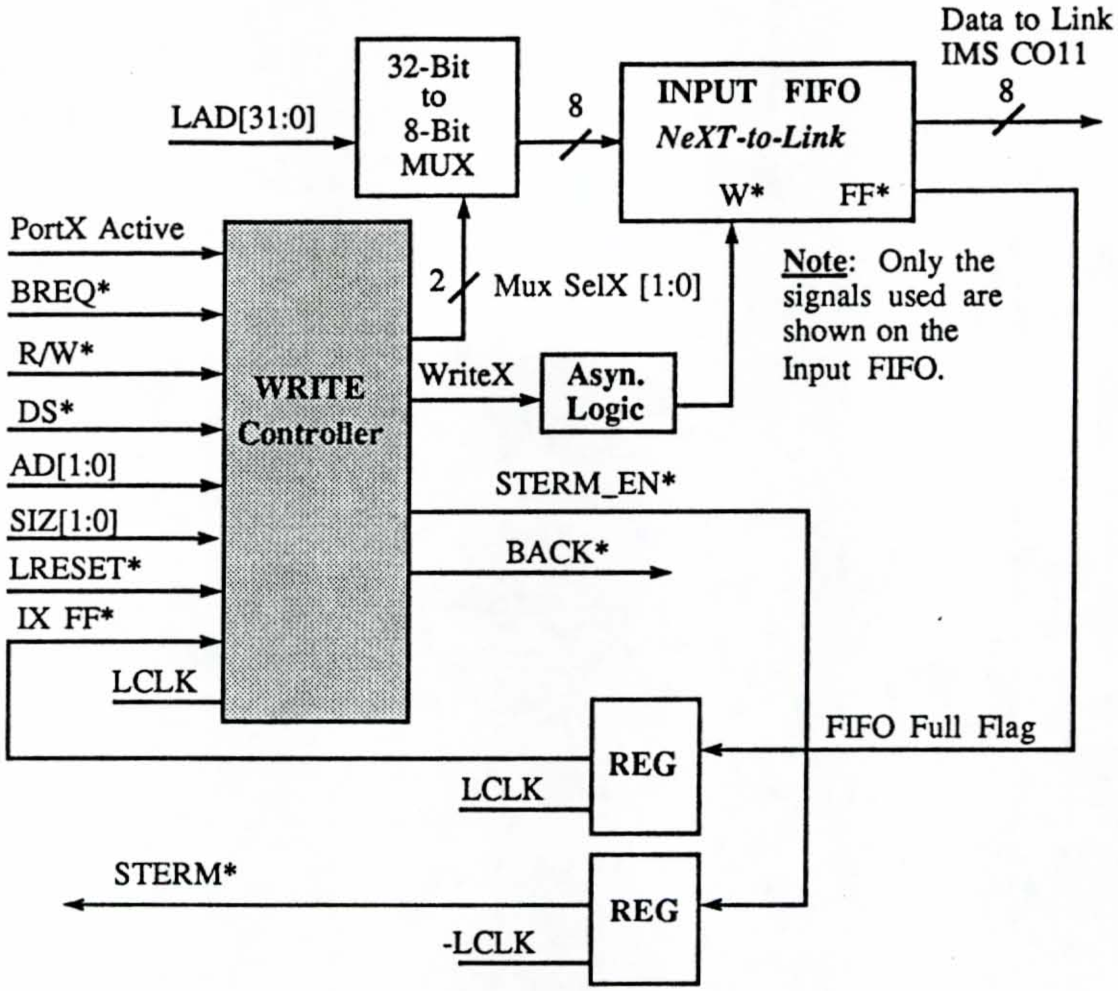
- O This controller is a composed of the following three controllers:
  - *(1) Write Controller*
  - *(2) Read Controller*
  - *(3) Status Controller*
- O Write Controller is activated for all write transactions between NBIC and Input FIFO's.
- O Read Controller is activated for all read transactions between the Output FIFO's and the NBIC.
- O Status Controller is responsible for placing one out of seven status flags (per port) on the lsb of the address line.

Top-level Block Diagram of Input Controller



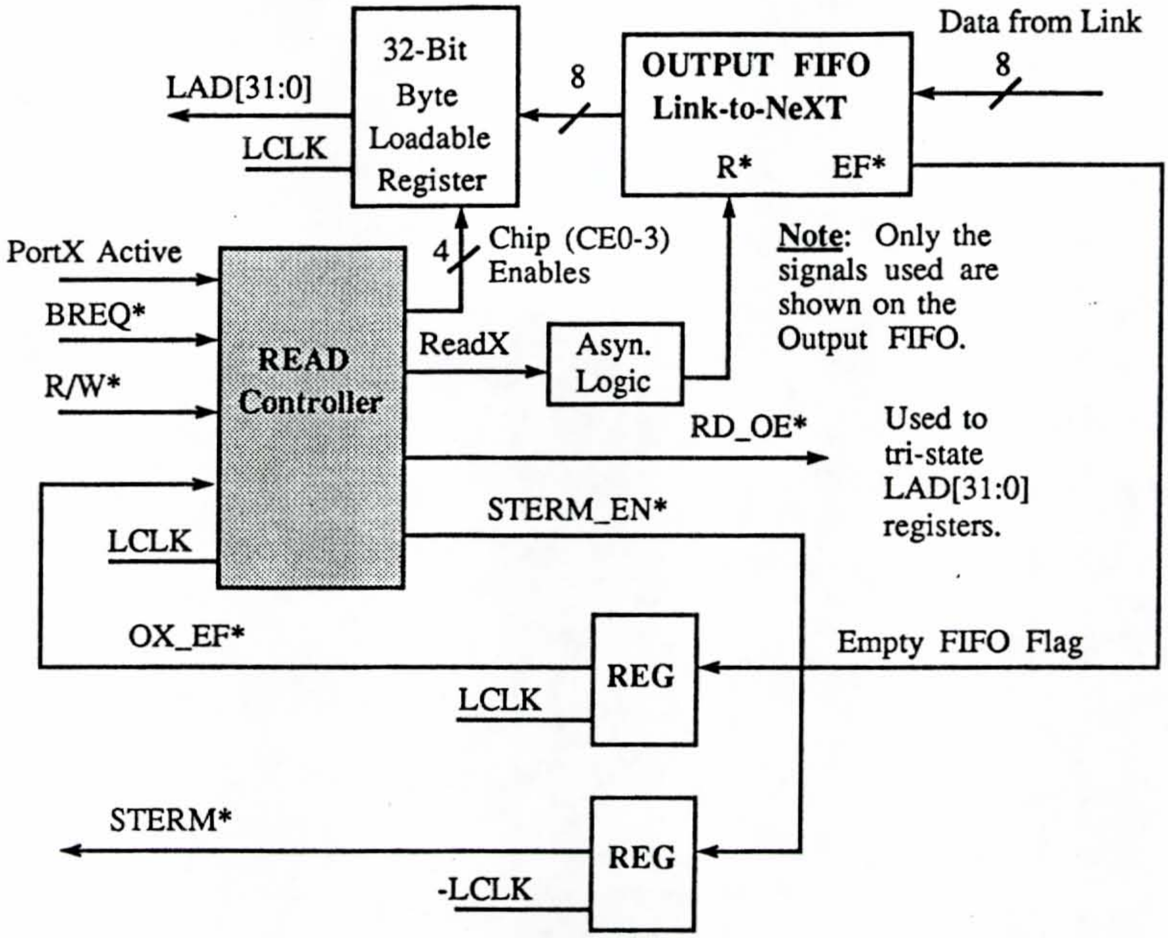
Where X is the Port Number (0-3).

Top-level Block Diagram of
Write Controller and its Interconnections



Top-level Block Diagram of

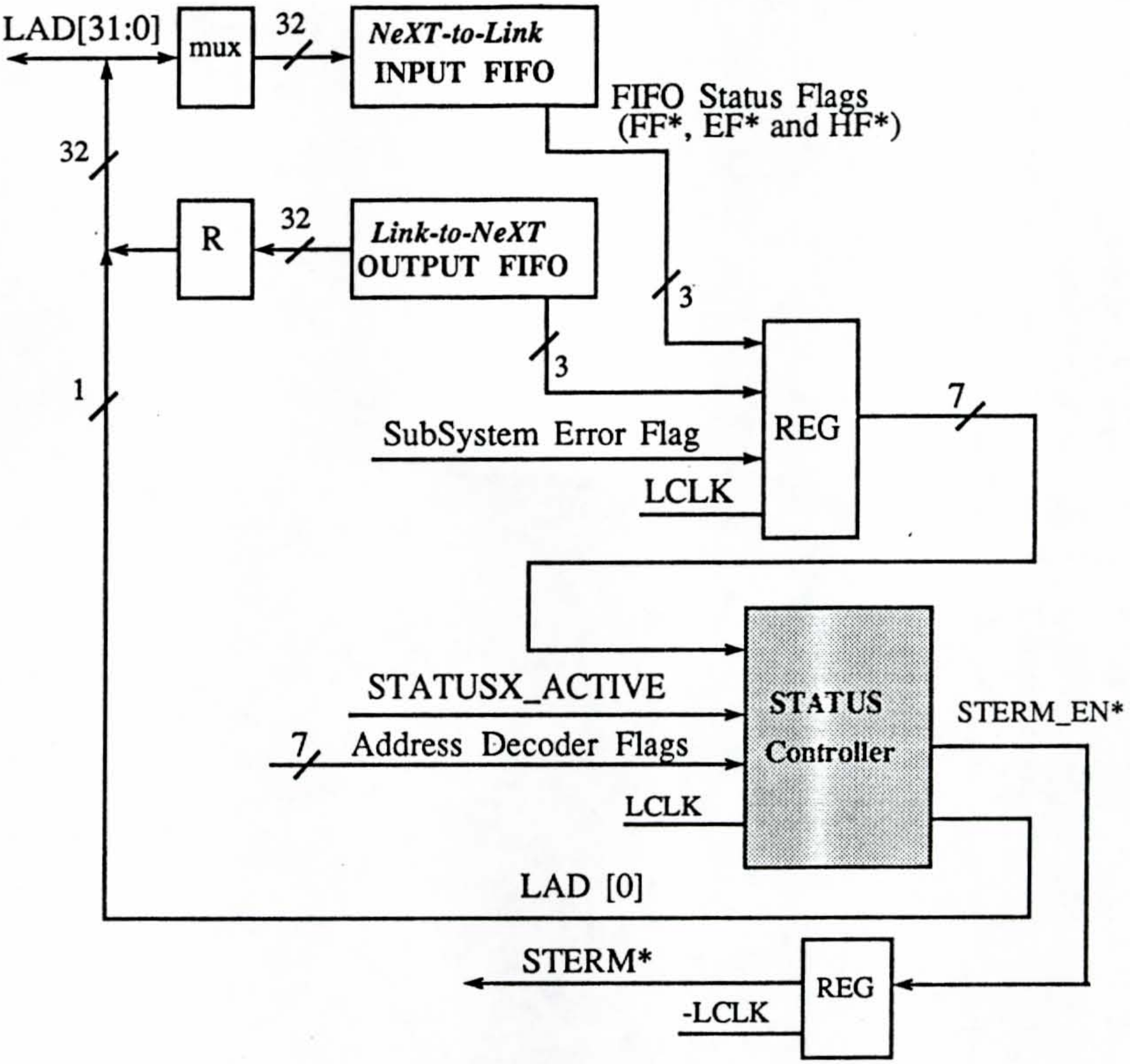
Read Controller and its Interconnections



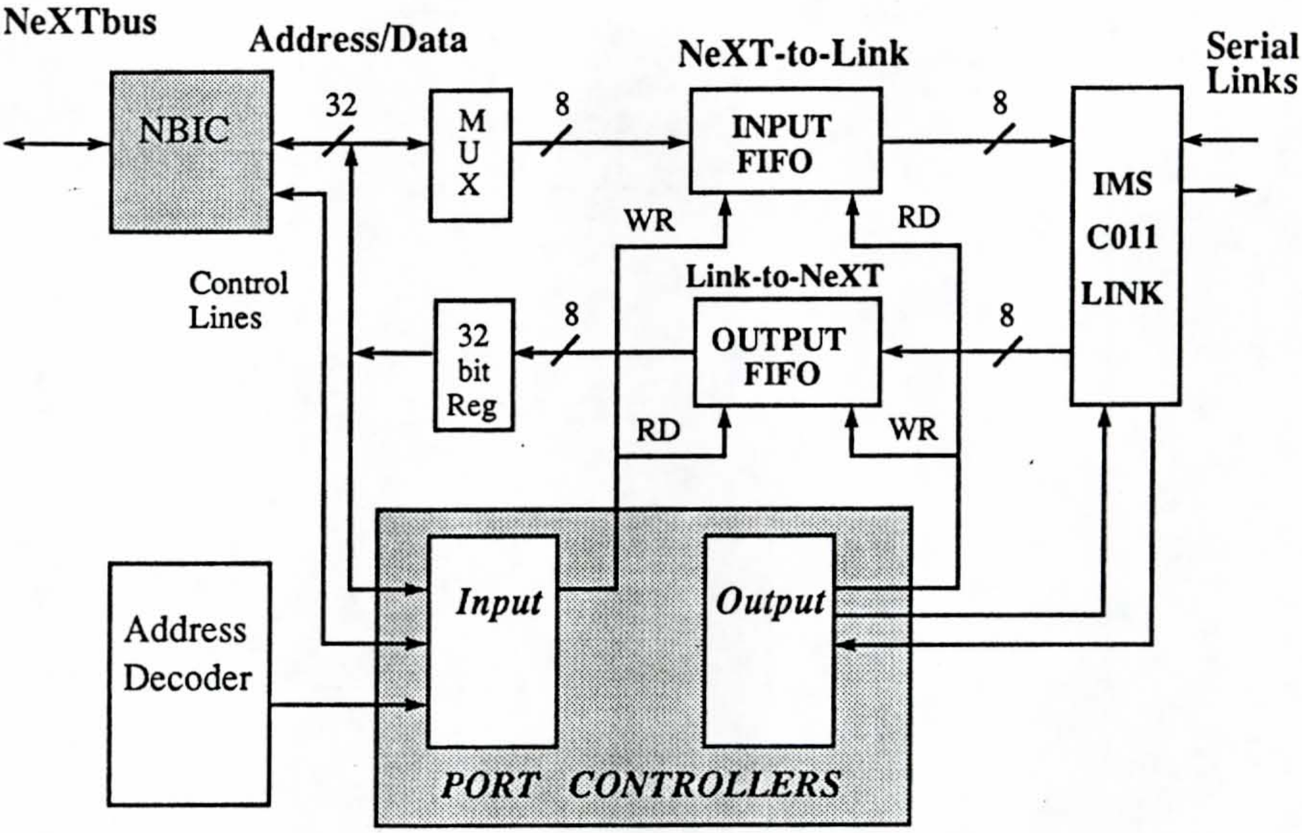
Note: Only the signals used are shown on the Output FIFO.

Used to tri-state LAD[31:0] registers.

Top-level Block Diagram of Status Hardware



Block Diagram of Port Controller Interconnections

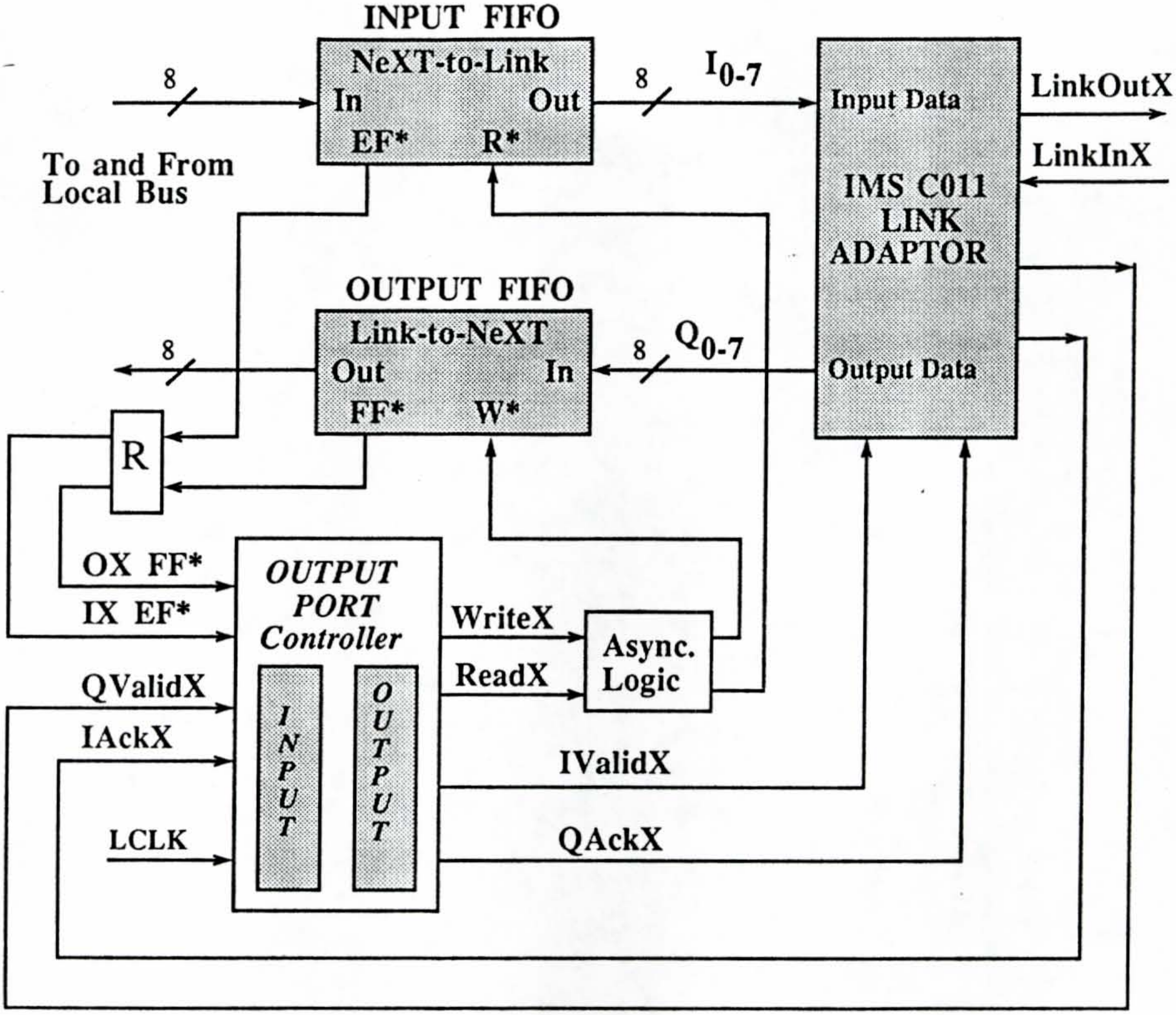


OUTPUT PORT CONTROLLER

Purpose: Responsible for all transactions between the Input / Output FIFO's and the Inmos Serial Link Adaptor IMS (C011).

- Link Adapter provides full duplex communication between the Output FIFO's and the Transputer serial links.
- This controller is composed of the following two controllers:
  - *(1) Input Sequence Controller*
  - *(2) Output Sequence Controller*
- Input sequence controller responsible for byte transfers from the Input FIFO to the IMS C011.
- Output sequence controller responsible for byte transfers from IMS C011 to Output FIFO.
- All transfers between FIFO's and IMS C011 are asynchronous via a two–wire handshake.

Top-level Block Diagram of Output Port Controller Interconnections

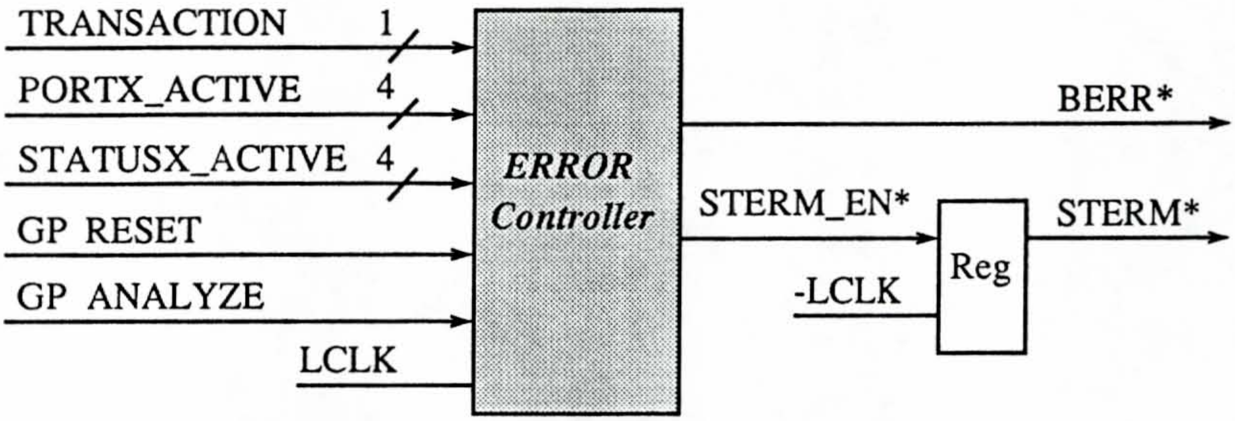


ERROR CONTROLLER

Purpose: Responsible for termination of an invalid transaction or acknowledgement for any Port Reset or Analyze transactions.

- O All inputs for controller received from Address Decoder.
- O Informs the user if an invalid memory address of the board has been referenced.
- O Verifies that a transaction does not have to wait till clock cycle 256 to terminate.
- O CPU board in NeXT computer terminates a transaction at clock cycle 256.

Top-Level Block Diagram of Error Controller.



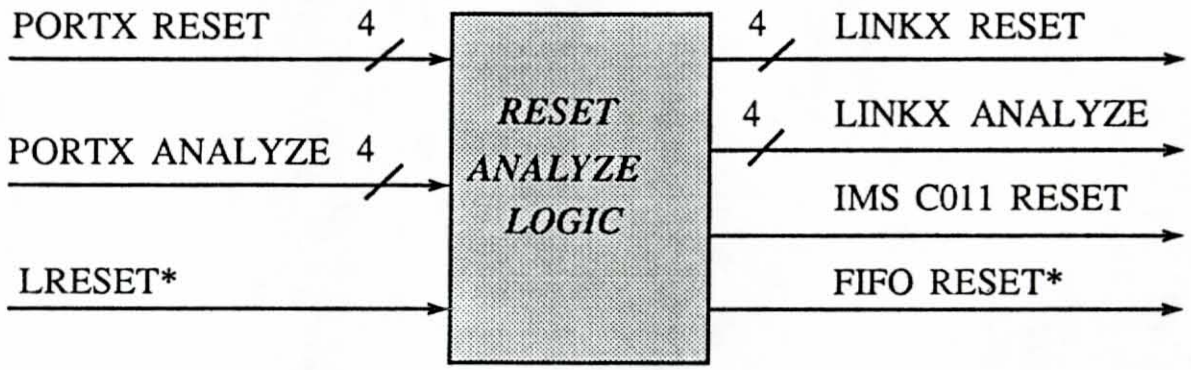
RESET, ANALYZE AND INTERRUPT LOGIC

- O Asserts the appropriate Port Reset signal when user selects to reset a port.
- O Asserts the appropriate Port Reset and Port Analyze signals when user selects to analyze a port.
- O The interface board shall assert its local interrupt signal (SINT*) when:

  (1) Data has been transferred from the coprocessing network to the interface board.

  (2) Any of the error flags from the coprocessing network are asserted.

Top-level Block Diagram of Reset and Analyze Logic



ADVANTAGES / DISADVANTAGES OF NBIC

Advantages

- Reduces amount of design time.
- Reduces amount of Logic on the board.
- Provides a simple local bus interface.
- Provides a single–chip interface to the NeXTbus.
- Provides store and forward capability.

Disadvantages

- Only supports a burst transfer size of 4 words.
- No error generation when a burst size greater than 4 is attempted.
- NBIC supports both Master and Slave boards; therefore, the NeXTbus Master/Local Slave Transaction FIFO is not used on the interface board.

WORK REMAINING

- Testability of the board needs to be designed.
- Interface software program needs to be developed. Program should perform the following:
  - *(1) Write (single–word or burst) known data patterns of arbitrary size into any of the four ports.*
  - *(2) Read data from any of the four ports.*
  - *(3) Report number of bytes compared and the total number of failures.*
- High–level simulation of the hardware modules.
- Detail design of interface board.
- Testing and debugging of the board.

TRAMs

- Off–the–shelf subassemblies which are composed of the following:
  - *(1) At least one processoor from the Inmos transputer family*
  - *(2) A few discrete components*
  - *(3) External memory*
  - *(4) Some contain application specific circuitry*
- All of the above components are on a single circuit board.
- TRAMs have a standard pinout (16 pins) and size.
- Basic size of a TRAM is called a SIZE 1 (Dimensions are 1.05" by 3.66").
- Since TRAMs have a standard pinout and size, it is easy to build customized motherboards for them.

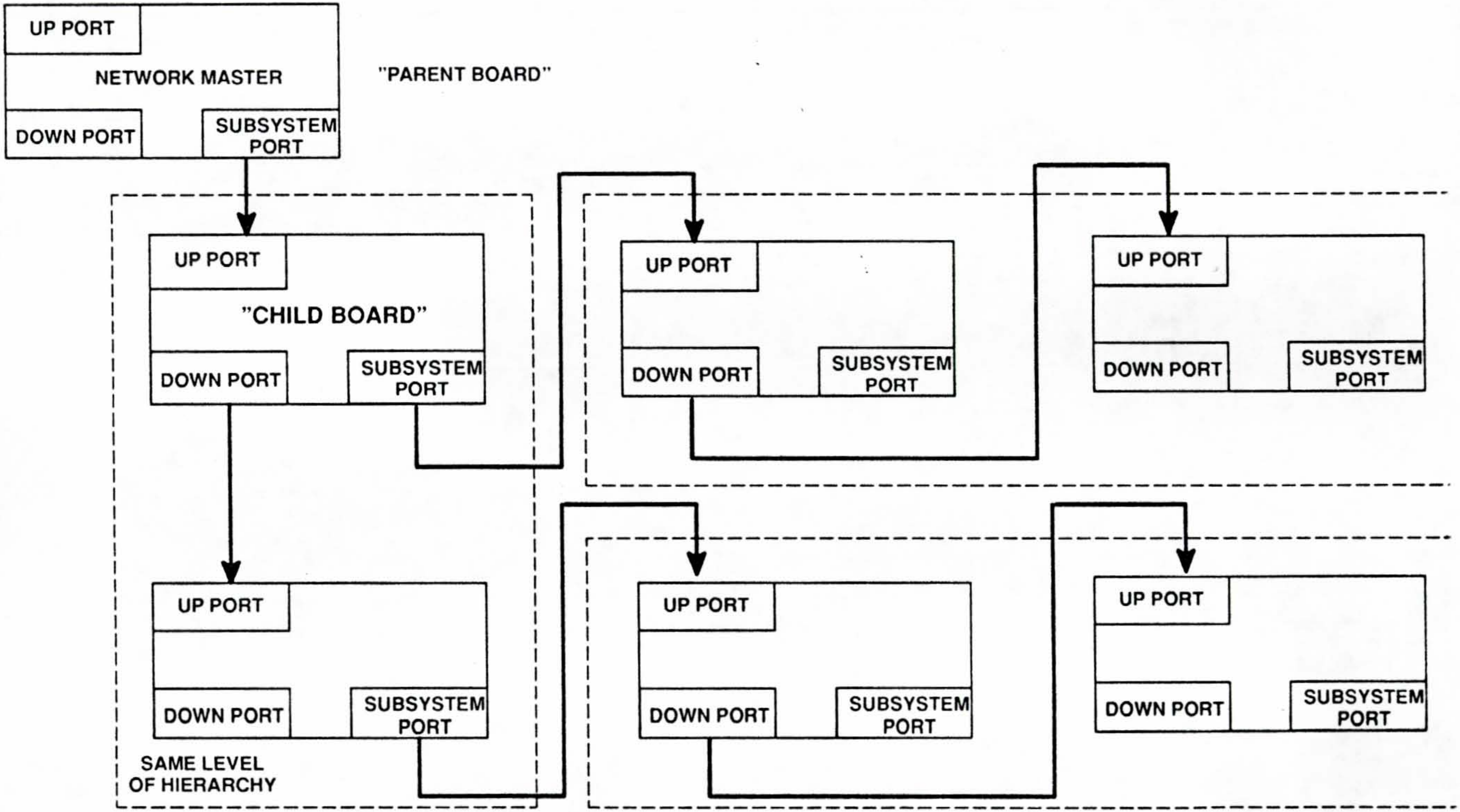
MOTHERBOARD HIERARCHICAL CONTROL

- Motherboards contain three ports to provide control of a network.
- The three ports are:
  - *(1) Subsystem Port*
  - *(2) Up Port*
  - *(3) Down Port*
- Subsystem Port is used by a master board to control other board(s).
- Boards are interconnected by connecting the Down Port of one board to Up Port of another board.

CONTROL SIGNALS PER PORT

- Each port has three active low signals:
  - *(1) not Reset*
  - *(2) not Analyze*
  - *(3) not Error*
- Reset and Analyze signals are asserted to either reset or analyze boards in the network.
- The Down and Subsystem Ports assert the "Reset" and "Analyze" signals.
- The Error signal is asserted by a motherboard when an invalid operation takes place.
- The Up Port receives the "Reset" and "Analyze" signals from its parent board.
- Status of the "Error" signal is feedback to the parent board from the Up Port of the child board.

SEVERAL MOTHERBOARDS INTERCONNECTED TO FORM A NETWORK OF COPROCESSORS



EVALUATION BOARDS DEVELOPED BY INMOS

(1) IMS B008 – *Supports a maximum of 10 SIZE 1 TRAMs*

(2) IMS B012 – *Supports a maximum of 16 SIZE 1 TRAMs*

Appendix II

Research Report by
Andres Alverez
on
Interfacing Transputers
to the
NeXTbus
in
Partial Fulfillment
of the
Requirements
for the
Degree of
Master of Science

Harry, 9:00 Tues Nov-20,
Brian

INTERFACING AN ARRAY OF COPROCESSORS TO THE NEXTBUS

BY

ANDRES ALVAREZ
B.S., University of Florida, 1986

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1990

ACKNOWLEDGEMENTS

I would like to take this time to thank my wife Jackie for all of her patience, motivation and understanding during all the months I spent working on my research report. Also, I would like to thank Dr. B. Petrasko for giving me the opportunity to work on the design of the interface board. Finally, I would like to thank my entire family for their support and motivation throughout the years I spent working on my graduate degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The first microprocessor introduced in the market was the 4-bit Intel 4004 in 1971. Since the development of this 4-bit processor, over 100 microprocessors have been introduced by U.S. manufactures. The development of the 32-bit microprocessors in the late 1970's has also introduced new bus architectures to exploit the high performance features of these processors. For example, Motorola developed its own bus architecture known as the Versabus in 1979 to support its 32-bit 68000 microprocessor. Since the development of the Versabus, a wide variety of bus architectures such as Versa Module Eurocard (VME) bus, Multibus, Multibus II and Nubus have been introduced into the market (Stone, 1983).

Bus architectures are used to allow processors to communicate with other processors, peripherals or an array of coprocessors. The bus is composed of a set of signal lines which can be divided into three groups: address and data, data transfer control and arbitration. The address and data lines are used to carry the information which is to be communicated between two modules. The data transfer control lines are used to verify that the information is transferred during a bus transaction. Finally, the arbitration lines are used to guarantee that only one processor shall have ownership of the bus at any given time. With the wide variety of bus architectures developed, there is a variation in the exact number of signal lines in each group. Buses can be classified as either asynchronous or synchronous, multiplexed or non-multiplexed and as multiprocessing (W. D. Peterson, "VMEbus Handbook", 1989).

Asynchronous buses use completely responsive handshaking signals to coordinate data transfers between different modules on the bus. The handshaking lines are used to indicate the beginning of valid data on the bus and the reception of the data by the receiving module. The data transfer rate is the data rate of the slower module participating in the transaction. Synchronous buses have a common clock on the backplane to coordinate all data transfer. Data transfers occur on the rising or trailing edge of the clock with one transfer per clock period. Some synchronous bus architectures provide a means of flow control. Flow control is a method in which the transmitting or receiving module can control the data rate on the bus.

In a multiplexed bus architecture, the address and data lines share a common set of pins on the backplane. A two bus cycle is required to transfer data since the address information is transferred during the first cycle and data on the second cycle. In a non-multiplexed bus architecture, the data and address lines have a different set of backplane pins. This type of configuration requires only one bus cycle to transfer data but more signal lines exist on the backplane.

Multiprocessing bus architectures allow more than one module to initiate bus transactions. A module that can initiate a data transfer bus cycles is known as a bus Master. A bus Master can also act as a receiving module when it is addressed by another bus Master. The modules which cannot initiate bus cycles are know as Slaves. A Slave module can only participate in a bus cycle when it is addressed by a bus Master. When there are two or more bus Masters on a bus, an arbitration cycle is initiated to determine which bus Master shall have ownership of the bus. The protocol used by buses to determine the bus Master is different between bus architectures. Some buses use a strictly fair arbitration technique known as Round-Robin while others provide a scheme which consists of different levels of priority.

There are a wide variety of bus architectures in the market today. The choice of which bus architecture to select depends on the type of application and data rate required by the user. Several common bus architectures and their corresponding features are shown in Table 1.1. Many of the bus architectures shown in Table 1.1 are widely used in different types of applications from research to development environments.

Table 1.1. Bus Architectures and their Features.

Bus Description	*Sync/ Asyn*	*Multi-plexed*	*Data Widths*	*Address Widths*	*Multipro-cessing*	*Governing Body*
IBM PC	Async	NO	8	20	NO	IBM
Multibus	Async	NO	8, 16	8,16,20,24	YES	IEEE 796
Multibus II	Sync	YES	8,16,20,24	32	YES	Intel
Nubus *	Sync	YES	32	32	YES	TI
NeXTbus *	Sync	YES	32	32	YES	NeXT
Q-bus	Async	YES	8, 16	16, 18, 22	YES	DEC
VME	Async	NO	8,16,24,32	16, 24, 32	YES	IEEE 1014
SCSI bus *	Both	YES	8	8	YES	ANSI

SOURCE: W. D. Peterson, The VMEbus Handbook: A User's Guide to the IEEE 1014 and IEC 821 Microcomputer Bus (VITA Publications, 1989), pg. 5, Table 1-1.
* These entries were compiled by author of this research report.

Most bus architectures have a corresponding chip set to aid in the interfacing of modules. For example, the Nubus architecture by Texas Instruments has a bipolar chip set which provides a full 32-bit Nubus Master/Slave interface. The chips are the SN74ACT240 controller and the SN74BCT2420 address/data transceiver. The NeXTbus, which is a superset of the Nubus, has the NeXTbus Interface Chip (NBIC) to aid designers in interfacing modules to the NeXT computer. The NBIC is a 144-pin CMOS Very Large Scale Integration (VLSI) chip which connects directly to the NeXTbus and provides an interface to a local bus (Mikkelsen, "NBIC", 1989).

The Nubus architecture is a synchronous (10 MHz) multiplexed bus which requires high current transistor-to-transistor logic (TTL) drivers and termination on all signal lines as defined by the IEEE standard 1196. The differences between the Nubus and NeXTbus architecture are the clocks, data transfer rate, electrical interface and number of arbitration cycles. The NeXTbus uses a 12.5 MHz clock, low power CMOS drivers and does not require termination on external signal lines. Since the NeXTbus architecture does not require external termination, a single-chip interface implementation is available (Mikkelsen, "NeXTbus", 1989).

The purpose of this research report is to demonstrate the use of the NBIC to interface an array of coprocessors, which shall be used in an operational and development environment, to the NeXTbus. The report shall provide a functional top-level design for a Host-to-Motherboard interface board. The board shall be located in the backplane of the NeXT cube and the Motherboard shall be part of an array of coprocessors. The Motherboard shall contain serial link(s) which shall be used to receive or transmit information to the Host-to-Motherboard board located in the NeXT cube. The Host-to-Motherboard board is a parallel-to-serial and serial-to-parallel link interface on a standard NeXTbus card. The board shall interface with the array of coprocessors via serial links versus a shared memory approach. In a shared memory approach, the NBIC data bus is directly connected to the data bus of the coprocessors. Instead, the information on the NBIC data bus is serially transmitted via Inmos link adaptors such as the IMS C011 and C012 to the serial links of the Motherboard located in the array of coprocessors. The different modules on the interface board shall be described in a design language known as AHPL (a Hardware Programming Language). The description of this language for readers not familiar with its syntax can be found in the two references given for Hill and Peterson. To supplement the AHPL description for the functional modules of the board, state-diagrams and timing diagrams shall be supplied to aid the designer in the detailed design of the board.

CHAPTER II

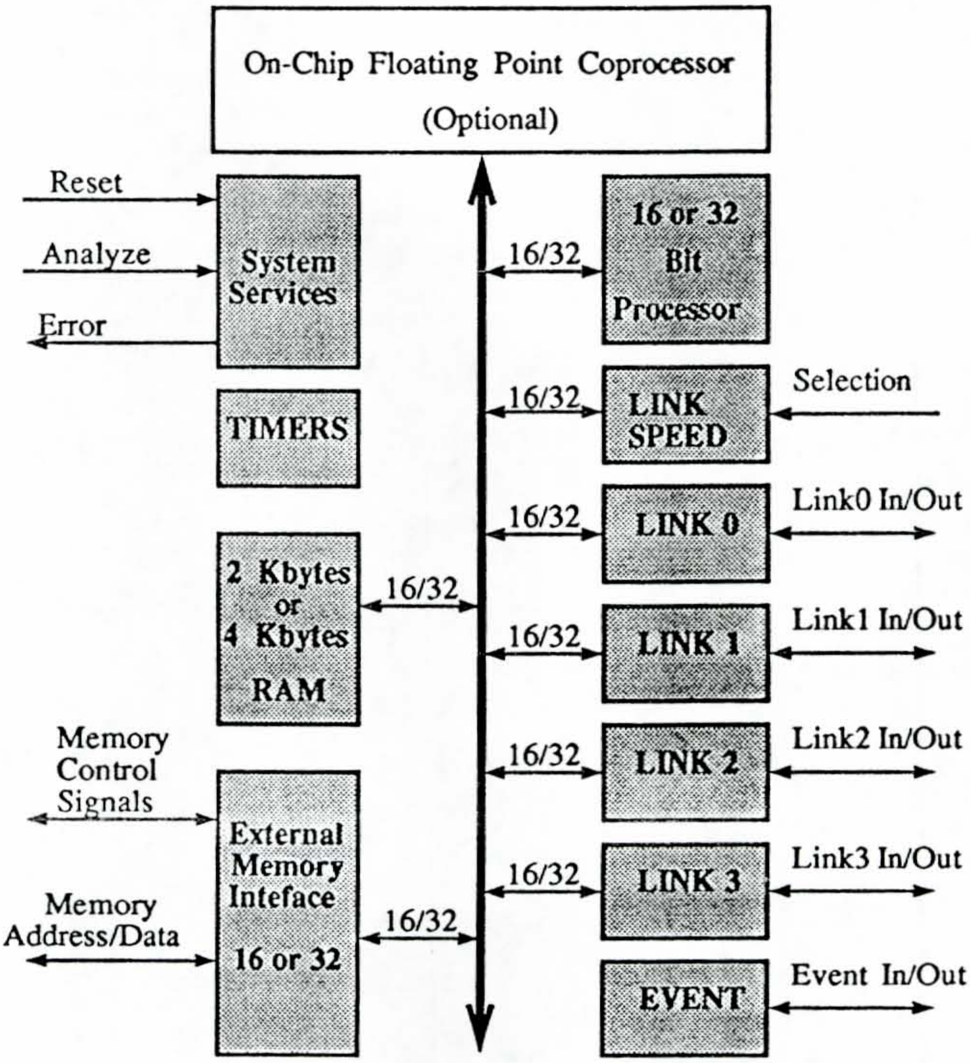
COPROCESSOR ARCHITECTURE

Introduction to the Inmos Transputer

The architecture for the array of coprocessors shall consists of a combination of Inmos 16 and/or 32-bit transputers, dual-inline transputer modules (TRAM's) and the IMS C004 32 way link switch. By mixing and matching the above components, designers can configure Motherboards which shall consists of large numbers of transputers to form powerful network of coprocessors. Motherboards allow control and interconnection of a number of transputer modules to form the building blocks of an array of coprocessors. They have a generic architecture and allow transputers to be accessed from a number of different host machines (Inmos, "Transputer Development and iq Systems Databook", 1989).

The Inmos transputer is a high-performance 32-bit microprocessor which is ideal for selected applications. The architecture of a transputer contains the following components: its own local memory; 4-high speed serial links; a 16 or 32 bit processor; a 16 or 32 external memory interface; internal timers for real-time processing; system services; an external event interrupt and some contain an on-chip floating point processor. A block diagram of the transputer architecture is shown in Figure 2.1. This programmable VLSI transputer chip was designed specifically to function as a component processor in a network of array coprocessors. Processes to be executed are queued by a hardware scheduler in a Round-Robin fashion with two priority levels. Processes which are waiting for internal or external communication are descheduled and later rescheduled at the appropriate time by the scheduler. The transputer can either boot from Read Only Memory (ROM) or from any of its four links. The next section shall explain how the transputer is bootstrapped from link and how a user can peek (READ) or poke (WRITE) into the addressable address space of the transputer (Inmos, "The Transputer Databook", 1989).

Figure 2.1. Block Diagram of Inmos Transputer Architecture.



SOURCE: Inmos, The Transputer Databook 2ed. (Redwood Burn LTD, Trowbridge : 1989), pg 48, Figure 1.1.

Boostrapping the Transputer

After the trailing edge of the RESET pulse, the transputer begins its initialization sequence by executing the memory configuration routine. After the memory configuration routine is complete, then the bootstrap routine is started. The transputer can either be booted from ROM or link depending on the state of the signal BootfromRom on the transputer. If BootfromRom is high, then the transputer shall begin to execute instructions in ROM. Usually, there will be a jump instruction in the ROM to a defined location in external memory. External memory shall contain the required data to bootstrap the transputer. If the signal BootfromRom is low, then the transputer shall wait for a control byte to arrive on any of its four serial links. Any location in either internal or external memory can be read or written while the transputer is waiting to be bootstrapped from link. There are three cases that can occur depending on the value of the control byte. (Inmos, "Transputer Databook, 1989").

If the control byte equals zero, then the POKE (write) operation is executed. After a control byte equal to zero is received, the transputer expects eight more bytes for a 32-bit architecture or four more bytes for a 16-bit architecture. For the 32-bit architecture case, the first four bytes after the control byte represent the internal or external address (least significant byte first). The last four bytes represent the data. The data is written into the address and the link shall wait for the next control bye. The address and data must be received on the same link as the control byte.

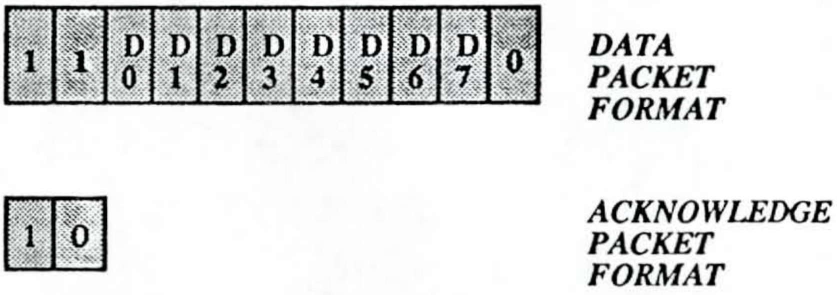
If the control byte equals one, then the PEEK (read) operation is executed. After the transputer receives a control byte equal to one, it shall expect four more bytes for a 32-bit architecture and two more bytes for a 16-bit architecture. The bytes received after the control byte represent the internal or external address at which to PEEK. The word stored at the given address is then transmitted on the same link as the control byte.

While the transputer is waiting to be boostrapped, any number of POKE or PEEK operations can take place. Any of the four serial links can be used to POKE or PEEK. The only restriction is that all information following the control byte must be transmitted on the same link as the control byte. Once the control byte is greater than one, then the transputer shall begin to load its bootstrap program from link. The control byte represents the number of bytes to follow. The transputer shall continue to read data on the same link as the control byte until all the bytes have been read. Once all the bytes have been read, the transputer shall begin to execute the program starting at a defined location in memory denoted by MEMSTART.

Link Communication Protocol

The serial links on the transputer are used to create a network of processors of arbitrary size and topology. The main advantage of using point-to-point links versus a multiprocessing bus architecture is that no control technique is required to access the medium. Medium access control algorithms such as Carrier Sense Multiple Access with Collision Detection (CSMA/CD) or the IBM Token Ring are required on multiprocessing buses. The links are dedicated channels between two transputers or between a transputer and an external device. The link interface consists of two unidirectional lines each which can transmit or receive data or control information. The communication protocol for the data packet transmitted on a link consists of a start bit followed by a one bit followed by eight data bits followed by a stop bit. Each data packet must be acknowledged before the next byte is transmitted. The acknowledge packet protocol consists of a start bit followed by a stop bit. The packet can be acknowledged before the entire byte has been received. This indicates to the transmitting transputer that there is room to buffer the current byte and that there is room for another byte to be transmitted. Link performance is improved by allowing the data packet to be acknowledged before the entire packet is transmitted. Links are used to simplify the development of system configuration. Communication between transputers can be achieved as long as the link speed is the same on both transputers. The transputers which are communicating via link do not need to have a common clock. Link communication is not sensitive to clock phase. All Inmos transputers support a communication data rate of 10 Megabits (Mbits) per second. Several transputers have select lines where the user can select 5, 10 or 20 Mbits per second communication rate. Figure 2.2 shows the formats for the data and acknowledge packets.

Figure 2.2. Link Data and Acknowledge Packet Formats.



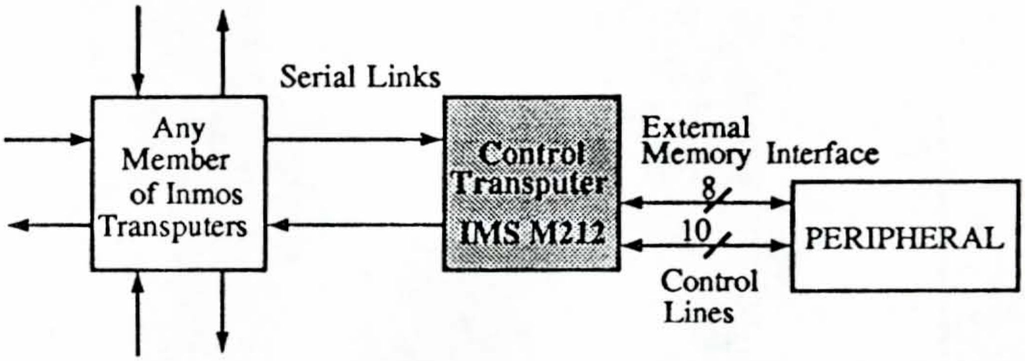
SOURCE: Inmos, The Transputer Databook 2ed. (Redwood Burn LTD, Trowbridge : 1989), pg 11, Figure 1.5.

Methods of Interfacing Peripherals to Transputers

The case shall arrive where a non-transputer device such as a disk drive, printer or a host needs to communicate with a member of the transputer family. Currently, there are three methods which can be used to interface transputers to peripherals. The three methods are by a specialized control transputer, by use of link adaptors and by external memory mapping.

Inmos has developed certain specialized transputers which can be used to interface specific type of peripherals to their transputers. The specialized control transputer shall then communicate to other transputers via its serial links. An example of specialized control transputer is the IMS M212. This transputer is used to interface a variety of disk drives to transputers. It contains two 8-bit bidirectional data ports and 10 dedicated control lines to interface the disk drive. The interface to the peripheral is implemented by specific hardware in the specialized control transputer. Figure 2.3 shows a top-level diagram of this configuration.

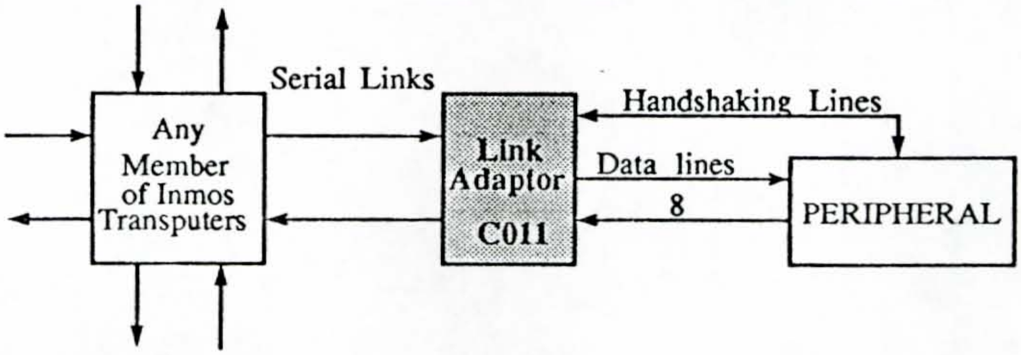
Figure 2.3. Peripheral Interfacing by a Control Transputer.


Serial Links
Any Member of Inmos Transputers
Control Transputer IMS M212
External Memory Interface
8
10
Control Lines
PERIPHERAL

SOURCE: Inmos, The Transputer Databook 2ed. (Redwood Burn LTD, Trowbridge : 1989), pg 26, Figure 5.1.

The second method is by using Inmos link adaptors such as the IMS C011 or the IMS C012. Link adaptors are used when a non-transputer device needs to communicate with a transputer. They contain two 8-bit data paths and handshaking lines, which are used to coordinate data transfers between the peripheral and the transputer. The link adaptor communicates with the transputer via serial links. The links of the link adaptor can be directly connected to the links of the transputer. Figure 2.4 shows a top-level diagram of this configuration.

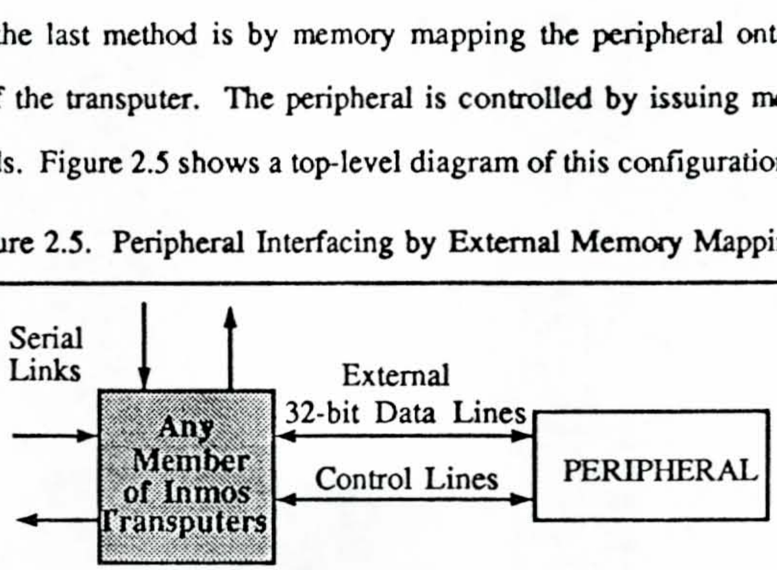
Figure 2.4 Peripheral Interfacing by a Link Adaptor.



SOURCE: Inmos, The Transputer Databook 2ed. (Redwood Burn LTD, Trowbridge : 1989), pg 26, Figure 5.2.

Finally, the last method is by memory mapping the peripheral onto the external memory bus of the transputer. The peripheral is controlled by issuing memory read or write commands. Figure 2.5 shows a top-level diagram of this configuration.

Figure 2.5. Peripheral Interfacing by External Memory Mapping.



SOURCE: Inmos, The Transputer Databook 2ed. (Redwood Burn LTD, Trowbridge : 1989), pg 26, Figure 5.1.

Inmos has developed several different configurations of their transputer architecture. To complete the introduction on the transputer, Table 2.1 shows the different transputers developed by Inmos and some of their features.

Table 2.1. Inmos Transputers and their Features.

Transputer	*16 or 32*	*ICT*	*On Chip RAM*	*External Memory*	*Resp. to Int.*	*Sustained Data Rate to*	
						External Memory	*Internal Memory*
IMS T805	32	33 ns	4 Kbytes	4 Gbytes	630 ns	40 MB/sec	120 MB/sec
IMS T801	32	33 ns	4 Kbytes	4 Gbytes	630 ns	60 MB/sec	120 MB/sec
IMS T800	32	33 ns	4 Kbytes	4 Gbytes	630 ns	40 MB/sec	120 MB/sec
IMS T425	32	33 ns	4 Kbytes	4 Gbytes	630 ns	40 MB/sec	120 MB/sec
IMS T414	32	50 ns	2 Kbytes	4 Gbytes	950 ns	26 MB/sec	80 MB/sec
IMS T222	16	50 ns	4 Kbytes	64 Kbytes	950 ns	20 MB/sec	40 MB/sec
IMS T225	16	33 ns	4 Kbytes	64 Kbytes	630 ns	30 MB/sec	60 MB/sec

NOTES: Data compiled from Inmos Transputer Data Book. Nomenclature for table.
Resp. to Int. = Response to Interrupts   ns = nanosecond
ICT = Internal Cycle Time   MB = Mega Bytes

Transputer Module Architecture

The transputer modules (TRAMs) are the next level of integration in an array of coprocessor. The transputer is the lowest level followed by the TRAM and the highest level being a Motherboard which contains both. TRAMs were developed by Inmos to satisfy the wide variety of configurations and applications which were in demand at the time that the transputer was developed. They are off the shelf subassemblies which are composed of at least one processor from the Inmos transputer family, a few discrete components, external memory and some contain application specific circuitry all on a printed circuit board. Inmos has developed a wide variety of TRAMs so designers can configure their own parallel processing systems for their unique application. They are available in different physical sizes with different memory sizes and with different functions. TRAMs have a standard 16-pin pinout and interface to each other via serial link. The basic size of TRAM is called a SIZE 1 and its dimensions are 1.05" by 3.66". Larger TRAMs exists and their dimensions can be up to 8.75" by 3.66" (SIZE 8). Since TRAMs have a standard pinout and size, it is easy to build customize Motherboards with sockets for them. Inmos has developed several evaluation boards, such as the IMS B008 and B012, specifically for TRAMs. The IMS B008 supports a maximum of 10 SIZE 1 TRAMS and the IMS B012 supports a maximum of 16 SIZE 1 TRAMS. Table 2.2 shows a list of the available TRAMs available from Inmos along with their corresponding features. (Inmos, "Transputer Development and iq Systems Databook", 1989).

TRAMs need to have the capability to control a network of transputer and/or more TRAMs. A network which is controlled by a master module is know as a subsystem of the master. The master module in the network controls the slave modules through a subsystem port. The subsystem port consists of three signals: subsystem-Reset, subsystemAnalyze and subsystemError. The Motherboard in an array of coprocessors shall have a master TRAM for controlling other TRAMs via its subsystem port.

The subSystemReset and subSystemAnalyze are used by the master module to reset and/or analyze its subsystem. The links of the transputer are active low during the reset phase of a network of coprocessors. SubSystemError is used by the master module to monitor the state of the error flag in its slave network. This signal is either an open-collector or open-drain signal so that all the error flags in the slave network can be wired "OR" together. The subsystem is controlled by reading or writing to an address in positive address space. The subsystem is reset or analyzed under the control of the transputer on the TRAM but must also be reset when the TRAM itself is being reset.

Table 2.2. Inmos TRAMs and their Features.

TRAM	*SZ*	*XP in TRAM*	*16 or 32*	*External SRAM*	*External DRAM*	*Subsystem Port*
IMS B416	1	T222	16	64 Kbytes	NONE	NO
IMS B401	1	T800 T425 T414	32	32 Kbytes	NONE	NO
IMS B411	1	T800 T425	32	NONE	1 Mbytes	NO
IMS B404	2	T800	32	28 Kybtes	2016 Kbytes	YES
IMS B417	4	T800 *	32	60 Kbytes	4032 Kbytes	YES
IMS B405	8	T800	32	NONE	8 Mbytes	YES
IMS B410	2	T801	32	156 Kbytes	NONE	NO

NOTES: Data compiled from The Transputer Development and iq Systems Databook. The variable SZ denotes the SIZE of the transputer. The symbol XP stands for transputer.

* Denotes 25 MHz transputer.

Generic Motherboard Architecture

Inmos has developed a generic motherboard architecture to aid in the development of powerful array coprocessors. A common motherboard architecture provides an easy way for system designers to build and configure systems consisting of several transputers and TRAMs. The architecture must allow different kinds of parallel processing networks such as a tree, cube or array to be configured. Also, each motherboard shall have the capability to be chained together with another motherboard in the network. All the link connections in the motherboard shall have the capability to be configured by software instead of being hardwired on the board. Finally, the motherboard shall have a standard hardware interface for a host machine to download the processes which are to be executed by the network. (Inmos, "Transputer Development and iq Systems Databook", 1989).

The generic motherboard architecture is composed of module slots for TRAMs which are interconnected via their links. Each module slot consists of two sockets which support a SIZE 1 TRAM. The number of TRAM slots depends on the size of the board. The first slot is denoted by slot 0 and the last slot is denoted by slot n. Inmos facilitates the interconnection of the links by developing a hardware link configuration standard and using the IMS C004 to configure the links via software. The links are configured in a pipeline fashion by connecting link 2 of slot 0 to link 1 of slot 1 continuing in this fashion all the way to slot n. Link 1 of the TRAM in slot 0 is denoted as the "Pipehead" and link 2 of the TRAM in slot n is denoted as the "Pipetail". The "Pipehead" and "Pipetail" are brought to an edge connector on the board. Both of these signals are used to interconnect all motherboards in a network in a pipeline fashion by connecting the "Pipetail" of one board to the "Pipehead" of another board.

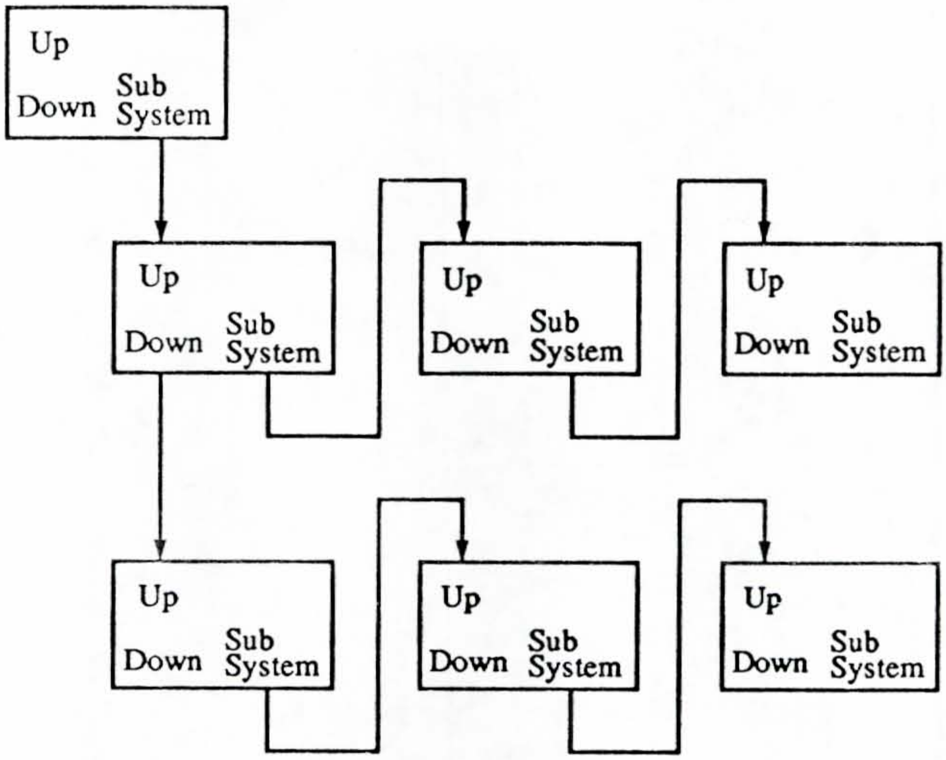
The remaining two links of each slot are brought out to an IMS C004 32-way link switch which is configured from software. The IMS C004 is a programmable link switch which directs any of the 32 link inputs to any of the 32 link outputs. The generic motherboard architecture does not specify how the links 0 and 3 are connected to the link switch. The designer of the motherboard can decide how links 0 and 3 are best connected to the link switch to serve their unique application. The only restriction stated by Inmos is that links 0 and 3 must be connected to the IMS C004. The number of link switches on the board depends on the number of slots on the board.

The IMS C004 link switch is controlled by T212 transputer. Each T212 transputer has the capability of controlling two link switches. Links 0 and 3 of each T212 is connected to the appropriate link switch. The remaining two links are used to configure the control transputers in a pipeline fashion for configuration. Link 1 of the first T212 transputer in the pipeline is connected to an edge connector on the board and it is denoted by the signal "ConfigUp". Link 2 of the last T212 transputer in the pipeline is also taken to an edge connector on the board and it is denoted by the signal "ConfigDown". The configuration pipeline of any two boards in a system can be interconnected by using these two signals. If a motherboard is the first board in the pipeline, then the configuration data shall come from one of the module slots on the board. In this case, a jumper is installed between the "Pipehead" and "ConfigUp" signals.

One of the specifications of a motherboard is that it must have the capability of providing hierarchical control of a network of transputers. This is accomplished on the motherboard by three ports: Up, Down and Subsytem. Each port consists of three active low signals which are all connected to an edge connector. The three signals are notReset, notAnalyze and notError. A motherboard can act as a network master by connecting its Subsystem port to the Up port of another board in the network. The boards in a subsytem are interconnected by connecting the Down port of one board to the Up port of another board. A board in a subsystem can act as a master to another board by its Subsytem port. Figure 2.6 shows several motherboards interconnected to form a network of processors.

The need shall arise where a motherboard requires an interface to a host machine such as the NeXT Computer. Inmos does not define any specific implementation for the Host interface. It does specify that the Host shall have access to the module pipeline, control a number of subsystems and have the capability of interrupting the Host when data in either direction has been completed. The Host machine shall interface to the module pipeline via slot 0 link 0 of the Motherboard. The purpose of this research report is to define and perform a top-level design of the Host-to-Motherboard link interface (slot 0 link 0) as stated in Chapter I.

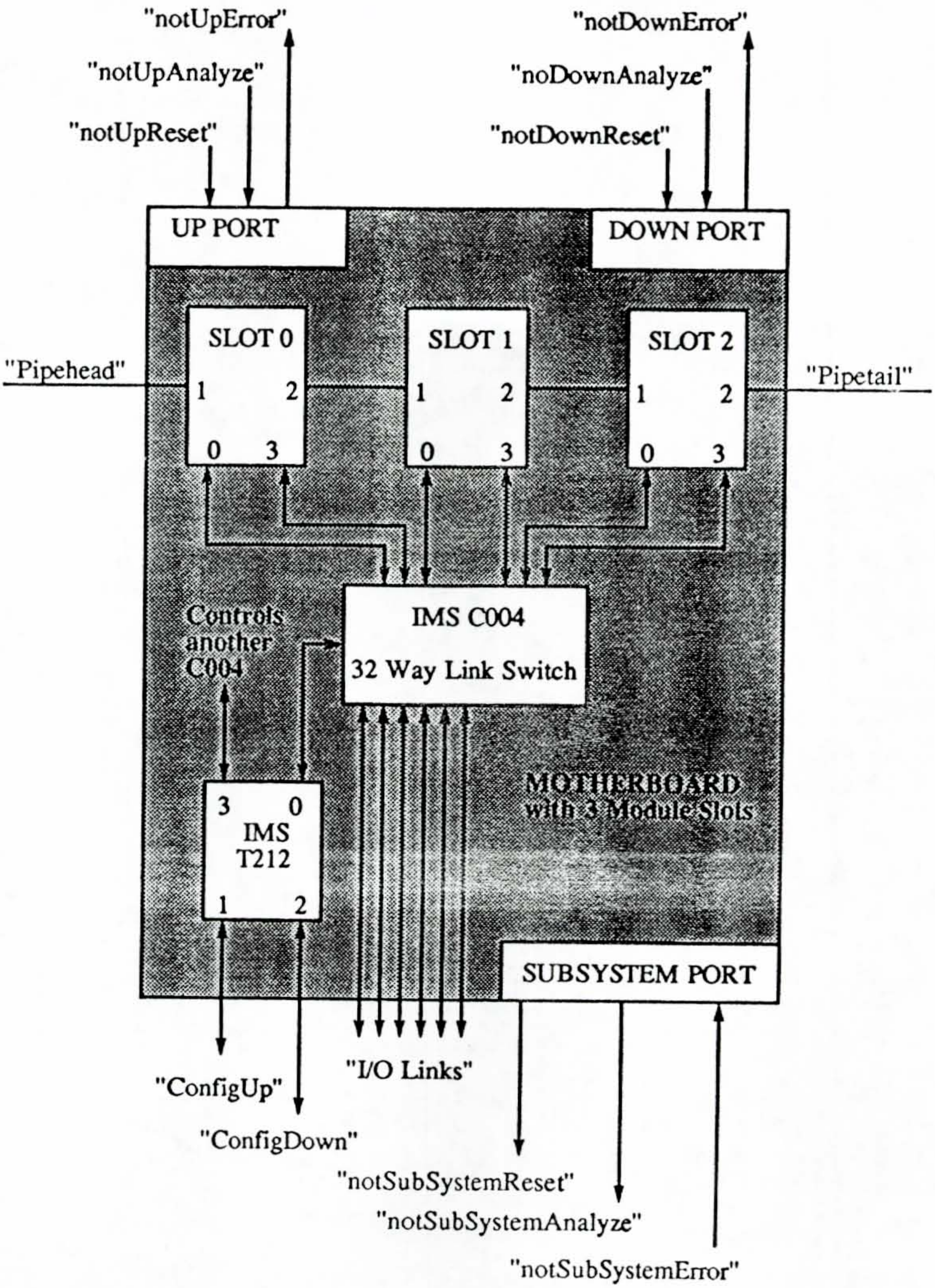
Figure 2.6. Motherboards Interconnected to Form a Network of Coprocessors.



SOURCE: Inmos, The Transputer Development and iq Systems Databook. (Redwood Burn LTD, Trowbridge : 1989), pg.215, Figure 9.7.

To conclude this chapter, Figure 2.7 shows a top-level diagram of a generic Motherboard architecture. The board shown in this figure contains three module slots, one IMS C004 link switch and a T212 configuration transputer.

Figure 2.7. Top-level Diagram of a Generic Motherboard Architecture.



SOURCE: Inmos, The Transputer Development and iq Systems Databook. (Redwood Burn LTD, Trowbridge : 1989), pg.213-215, Figures 9.4 - 9.6.

CHAPTER III

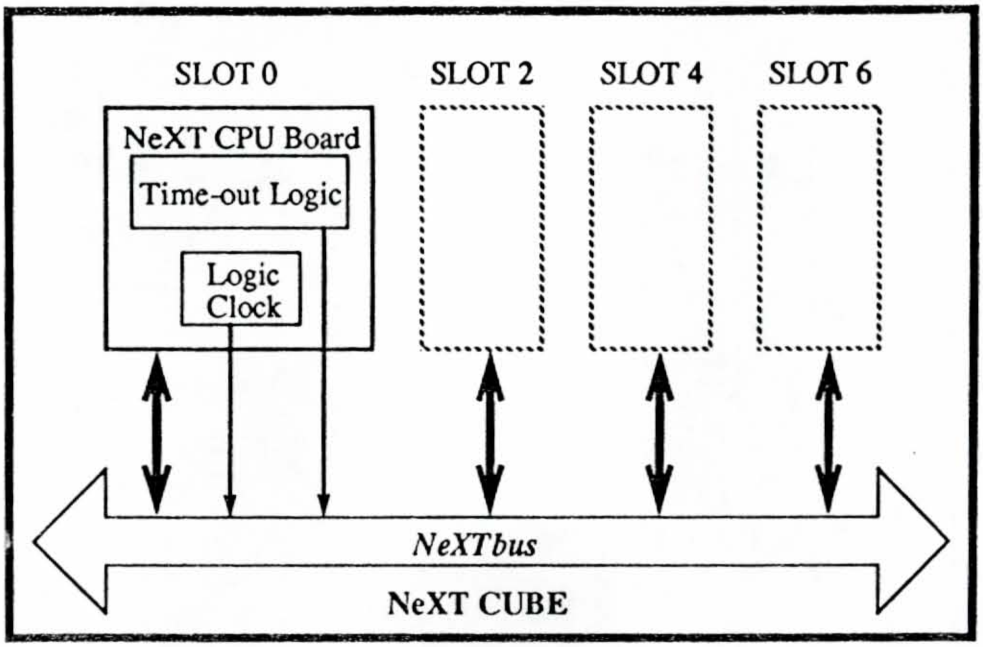
NEXTBUS ARCHITECTURE SPECIFICATIONS

Overview of the NeXTbus Features

This chapter shall introduce the communication protocol to interface with the NeXTbus. Only the relevant information about the NeXTbus, which shall be used to develop the functional design for a NeXT Computer to transputer Motherboard interface board shall be mentioned in this chapter. If the reader has any further specific interest on the NeXTbus architecture, all necessary information can be obtained from the "NeXTbus Specification" by Mikkelsen. All the information in this chapter is derived from the "NeXTbus Specification" by Mikkelsen.

The NeXTbus is a superset of the NuBus which was developed by Texas Instrument and is defined by the IEEE 1196 standard. The NeXTbus is a synchronous 12.5 MHz multiplexed bus which uses a strictly fair arbitration scheme. The bus is composed of 96 signal lines on the backplane, has 32 bit addressing which can support up to four GigaBytes (Gbytes) of address space, has a single chip interface via the NBIC and supports Master/Slave and Slave only boards. Flow control is available for both Master or Slave boards. Although the NeXT address space supports 15 slots, the NeXT cube only supports four boards. One of the four boards is the CPU board which is located in slot 0 and provides the 25 MHz backplane clock (BUSCLK) and time-out functions. The three remaining slots on the NeXT cube are slots 2,4 and 6. Figure 3.1 shows a block-diagram of the NeXTbus in the NeXT cube. The signals on the bus are classified into one of the following categories: utility, control, address/data, arbitration, parity and power. The different signals under each category shall be defined in this chapter.

Figure 3.1. Block Diagram of NeXTbus in the NeXT Cube.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification" (NeXT, 1989), Reorder Product #N6010, pg. 1-2, Figure 1-1.

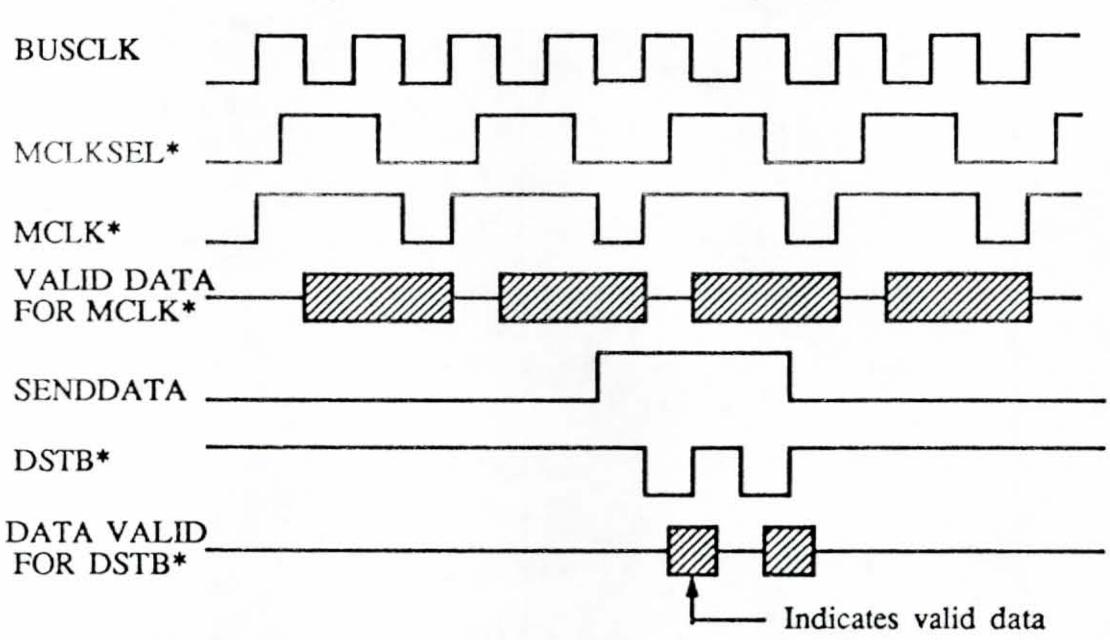
The addressing space of the bus is divided into 16 sections each consisting of 256 MegaBytes (MBytes). The addressing range for each slot is defined from s0000000 hex to sFFFFFFF hex where s is the slot id number of the board. The top 256 MBytes is further divided into 16 sections where each slot receives 16 MBytes. The 16 MBytes are called the slot address space and the address for each slot is defined from Fs000000 hex to FsFFFFFF hex. The slot address space is used for slot identification purposes and contains two words for interrupt control. The slot address space stores a 32-bit ID word across the top four words. The 16 most significant bits (msb's) represent a NeXT manufacturing code and the 16 least significant bits (lsb's) are allocated to board designers for ID purposes. The 32-bit words are divided across four 32-bit words to facilitate the use of byte-wide devices such as a ROM. Most of the slot address space shall be controlled by the NBIC. Each word in the slot address space occupies four address locations since each location contains a byte of information. Six words in each slot address space from FsFFFFE8 hex to Fs-FFFFFF hex are used to store the board's ID code and the two interrupt bytes. The two interrupt bytes only store a valid bit which is located in the msb of the byte. The interrupt byte in location FsFFFFE8 hex is a read only address and is used to interrupt the CPU. The other interrupt control byte is a mask bit which is stored in the msb of the byte in address FsFFFFEC hex. The mask bit is used to coordinate the interrupts on the bus since multiple boards may be requesting an interrupt from the CPU board.

NeXTbus Clocks

The CPU board in slot 0 provides the 25 MHz clock defined by BUSCLK and provides another clock denoted by MCLKSEL* (Major Clock Select). The asterisk on signal names indicates that the signal is an active low signal. This notation shall be used throughout the research report. MCLKSEL* shall select every other low phase of BUSCLK. Every board which shall be installed in the backplane of the NeXT cube needs to generate two internal clocks using the two clocks provided on the backplane. The two clocks are MCLK* and DSTB*. MCLK* provides a timing reference for control signals and for single word transactions on the bus. DSTB* provides a timing reference when burst transactions are being performed.

MCLK* is derived on the board by the logical "OR" of BUSCLK and MCLK-SEL*. This shall generate a clock signal with a period of 80 nanoseconds (ns) between the rising edge of the clock. The 80 ns between the rising edges is known as a major clock cycle. DSTB* is derived by the logical "NAND" of BUSCLK and an internal board signal called SENDDATA. The signal SENDDATA is asserted for two periods of BUSCLK and indicates that two words shall be transferred during the current major clock cycle. The module which is transmitting on the bus in burst mode shall assert the DSTB* clock along with the data. The receiving module during burst transfers shall latch data on the rising edge of DSTB*. The data is driven on the rising edge of both clocks (MCLK* and DSTB*) and sampled on the trailing edge of the clocks. Figure 3.2 shows a timing diagram of all the clocks used in the NeXTbus architecture along with their relationships to the address and data lines. The address/data lines on the bus are inverted such that a of value 0 hex equals FFFFFFFF hex and the value FFFFFFFF hex equals 0 hex.

Figure 3.2. NeXTbus Basic Timing Signals.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification" (NeXT, 1989) Reorder Product #N6010, pg.1.6 - 1.7. Figures 1.5 and 1.6.

NeXTbus Signal Description

The NeXTbus architecture is composed of 96 signal lines which are grouped into six categories based on the function they perform. The categories are utility, control, address/data, arbitration, parity and power. The 96 signal lines connect to a 96-pin Euro-DIN connector which must be on all NeXTbus boards. The pinout assignment of the signals on the NeXT backplane are shown in Appendix A. Table 3.1 shows all the signals and the category in which they are grouped in. A description of all the sig-

Table 3.1. NeXTbus Signals.

SIGNAL CATEGORY	DESCRIPTION	NO OF PINS
UTILITY		
RESET*	Reset Pulse	1
SID [3:0]	Slot ID	4
INT*	Interrupt	1
PON	Power On	1
PUP	Power Up	1
CONTROL		
BUSCLK	25 MHz Clock	1
MCLKSEL*	Major Clock Select	1
START*	Start	1
ACK*	Acknowledge	1
TM[1:0]	Transfer Mode	2
DSTB*	Data Strobe	1
DRQ*	Data Request	1
ARBITRATION		
ARB[3:0]*	Arbitration Number	4
RQST*	Bus Request	1
ADDRESS/DATA		
AD[31:0]	Address/data lines	32
PARITY		
SP[3:0]*	System Parity	4
SPV*	System Parity Valid	1
POWER		
		10
	+5V	14
	GND	4
	+12V	4
	-12V	
RESERVED		
RSVD[5:0]	Reserved Lines	6

SOURCE: Catherine Mikkelsen, "NeXTbus Specification" (NeXT, 1989), Reorder Product #N6010, pg. 2.2, Table 2.1.

nals is given below:

RESET* is an asynchronous signal used to reset all the boards on the backplane of the NeXT cube. It is pulled up by a 470 Ohm resistor since it is an open-drain signal. It is asserted by the NeXT power supply at initial powerup.

ID[3:0]* are the slot identification lines which are binary coded on the backplane as shown in Table 3.2. These signal lines are used during arbitration cycles.

INT* is a wired "OR" signal that is connected to every board on the backplane. NeXT does not specify any protocol for this signal. This signal can be used to allow Slave only boards a mechanism for interrupting the CPU board. This signal is pulled up by a 220 Ohm resistor on the backplane.

Table 3.2. Encoding of the Slot Identification Numbers.

	SID 3	SID 2	SID 1	SID 0
SLOT 0	GND	GND	GND	GND
SLOT 1	GND	GND	+5V	GND
SLOT 2	GND	+5V	GND	GND
SLOT 3	GND	+5V	+5V	GND

SOURCE: Catherine Mikkelsen, "NeXTbus Specification" (NeXT, 1989), Reorder Product #N6010, pg. 2.4.

PON is an active-high asynchronous signal which is asserted when the power supply has reached its stabilization point. This signal shall be deasserted before the power supply reaches a voltage level of 2.0 volts. This signal is pulled up in the NeXT power supply by a 470 Ohm resistor.

PUP is an active-high signal which is an input to the power supply circuity. When the CPU board asserts this signal between 2.4 and 5.0 volts, the power supply is ready to function. The power supply continues to function as long as the voltage stays 2.2 volts or greater. If the voltage drops below 1.0 volts for greater than 1 millisecond, then the power supply shall shut down.

BUSCLK is a 25 MHz signal provided by the CPU board in slot 0.

MCLKSEL* is a 12.5 MHz signal with a 75 percent duty cycle. This clock is also generated by the CPU board.

START* is asserted by the current bus master during the beginning of a transaction. It is asserted for one period and the type of transaction is encoded in the two lsb's of the address/data lines (AD[1:0]). When asserted, it lets the Slave module know that a valid address is on the bus.

ACK* is asserted during the acknowledge cycle by the Slave module to indicate reception of data or asserted at the beginning of an attention cycle. This signal is asserted for one clock period.

TM[1:0]* are the two transfer bits which serve several purpose during the different types of cycles. During a start cycle, these two bits along with the two lsb's of the address/data lines are driven by the bus master to encode the type of transaction to take place. During an acknowledge cycle, these two bits along with the two lsb's of the address/data lines are driven by the Slave module to encode the type of acknowledgment. During burst transactions, TM0* is used as a handshaking signal. During a burst write transaction, the bus Slave asserts TM0* to indicate that it can receive four more words. During a burst read transaction, the Slave module asserts TM0* when it places valid data on the data bus. This signal when used as an handshaking signals affects the following cycle or cycles. When a read or write burst operation begins, this signal is assumed to be asserted.

DSTB* is a data strobe signal which is asserted by the module which is currently transmitting data on the bus. It is asserted at the same time as data is placed on the bus.

DRQ* is a handshaking line that is asserted by the bus Master during burst transactions. When the bus Master is transmitting, it asserts this signal at the same time it places data on the bus. During a burst read operation, the bus Master asserts this signal when it can sink or receive four more data words. During a read transaction, the signal only affects the next cycle or cycles. At the beginning of a burst read transaction, this signal is assumed to asserted by the bus Slave. This signal is used for flow control.

ARB[3:0]* are open-drain signals which are used during the arbitration cycle to determine the next bus owner. At the end of the arbitration cycle, the winner's slot ID numbers are present on these signal lines. These signals are pulled-up on the NeXT backplane by a 220 Ohm resistors.

RQST* is an open-drain signal which is asserted by a bus Master who wants to contend for bus ownership. This signal can only be asserted if it is in the deasserted state.

AD[31:0]* are multiplexed bidirectional lines which carry either address or data. The contents of the AD bus are inverted as previously mentioned.

SP[3:0]* are the system parity lines. Each parity bit is associated with a corresponding byte. Parity bit SP[3]* correspond to the most significant byte AD[31:24]*, SP[2]* corresponds to AD[23:16]*, SP[1]* with AD[15:8]* and SP[0]* with AD[7:0]*. Each parity bit is asserted when the number of asserted data lines plus the parity bit is odd.

SPV* is asserted when parity bits have been generated for the valid data lines.

Different Types of Bus Cycles

There are three basic cycles which can occur on the NeXTbus. The type of cycle being performed is encoded by two signals denoted by START* and ACK*. The three basic types of cycles are start, acknowledge and attention.

The start cycle is encoded with START* asserted and ACK* deasserted. This is the first cycle in a transaction. During this cycle, the signals TM[1:0]* and AD[1:0] are driven by the bus Master to encode the type of transaction which is to take place.

Table 3.3 shows all the possible encoding schemes for the start cycle. There are 16 valid codes since 4-bits are used to encode the type of transaction to be performed.

The acknowledge cycle is the last cycle in a transaction. Its purpose is to provide status information of the current transaction to the bus Master. During this cycle, the bus Slave asserts ACK* and deasserts START*. The signals TM[1:0]* are driven by the bus Slave to encode the status information on the bus. Table 3.4 shows the four possible status information that can be encoded by the bus slave. A bus transfer complete status is returned by the Slave when the transaction completed normally. An error status is returned by the Slave when data received or transmitted may be erroneous. The bus Master can receive a time-out status which is generated by the CPU board. This indicates that no Slave responded to the address generated by the master. If no acknowledge cycle occurs after 256 clock cycles, then the CPU board generates an acknowledge cycle with a time-out status. The final status which can be encoded by a Slave module is a try again later status. This indicates to the bus master that the addressed Slave can not complete the transaction at this time. The Master can retry the transaction at a later time.

The last type of bus cycle is an attention cycle. Attention cycles consists of one major clock cycle and the Master asserts both START* and ACK* signals. There are two valid attention cycles. These two cycles are an attention-null and attention-resource lock. An attention-null cycle is used to restart an arbitration cycle This cycle is issued when a bus Master has ownership of the bus but does not engage in a transaction and another bus Master asserts RQST*. The current bus owner shall issue an attention-null cycle to begin a new arbitration cycle. The last attention cycle is an attention-resource lock. This cycle is issued by a bus Master to indicate to all Slave modules on the bus that a bus-locked transactions is occurring. Slave modules shall lock any resources such as memory that may interfere with a bus-locked transaction. At the end of a bus-locked transaction, the bus owner shall issue an atten-

Table 3.3. Valid Encoding Codes for Start Cycle.

TM1*	TM0*	AD1*	AD0*	TYPE OF CYCLE
L	L	L	L	Write Byte 3
L	L	L	H	Write Byte 2
L	L	H	L	Write Byte 1
L	L	H	H	Write Byte 0
L	H	L	L	Write Halfword 1
L	H	L	H	Write Burst
L	H	H	L	Write Halfword 0
L	H	H	H	Write Word
H	L	L	L	Read Byte 3
H	L	L	H	Read Byte 2
H	L	H	L	Read Byte 1
H	L	H	H	Read Byte 0
H	H	L	L	Read Halfword 1
H	H	L	H	Read Burst
H	H	H	L	Read Halfword 0
H	H	H	H	Read Word

SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg. 4-1, Table 4-1.

Table 3.4. Valid Encoding Codes for Acknowledge Cycle.

TM1*	TM0*	Type of Acknowledgment
L	L	Bus Transfer Complete
L	H	Error
H	L	Bus Time-out Error
H	H	Try Again Later

SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg.4-7, Table 4-4.

tion-null cycle to broadcast to all Slave modules to unlock there resources. Table 3.5 shows the valid encoding codes for an attention cycle.

Table 3.5. Valid Encoding Codes for Attention Cycle.

TM1*	TM0*	Type of Attention Cycle
L	L	Attention-Null
L	H	Reserved
H	L	Attention-Resource-Lock
H	H	Reserved

SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg.4-7, Table 4-4.

CHAPTER IV

NEXTBUS TRANSACTIONS

This chapter shall introduce the different data transactions available on the NeXTbus. The transactions covered in this chapter are single-word transfers, burst transfers and burst transfers with flow control. The timing diagrams in this chapter assume that a valid arbitration cycle has finished and a bus Master has ownership of the bus. The simplest type of transfers are single-word transfers. They are composed of a start cycle followed by an acknowledge cycle. The transfer mode bits (TM[1:0]*) and the two least significant bits (lsb's) of the address lines are used to encode the type of transfer and which bytes of the word are valid. A single-word read transaction begins by the bus Master asserting START* and encoding the transfer mode bits and address lines with the appropriate information. The transaction is completed when the addressed bus Slave asserts the signal ACK*, places the data on the bus and encodes the status of the transaction on the two transfer mode bits. Figure 4.1 shows a single-word NeXTbus read transaction. The sequence of events which are depicted on the timing diagram are as follows:

(1) D1: Denotes the first driving edge of the transaction. Bus Master asserts the signal START* and drives the address and transfer mode bit signal lines.

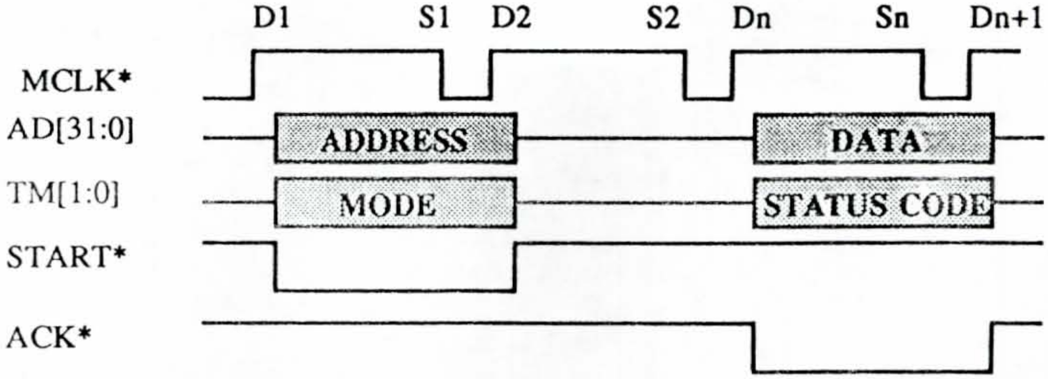
(2) S1: Denotes the first sampling edge of the transaction. During this edge, all bus modules sample the address and transfer mode bits lines to determine which module shall participate in a NeXTbus transaction.

(3) Dn: Denotes the last driving edge of the transaction. During this edge, the addressed bus Slave asserts the signal ACK*, places data on the bus and encodes the status information on the two transfer mode bits.

(4) Sn: Denotes the last sampling edge of the transaction. During this time, the bus Master samples the data and status information place on the bus by the slave module.

(5) Dn+1: First driving edge of the next transaction. The current or next bus owner must drive the signal ACK* to a determinate state.

Figure 4.1 Single-Word NeXTbus Read Transaction.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification" (NeXT, 1989), Reorder Product #N6010, pg. 4-9, Figure 4-6.

A single-word write transaction is very similar to a single-word read transaction. Figure 4.2 shows the timing diagram for a single-word write transaction. The sequence of events which are depicted on the timing diagram are as follows:

(1) D1: Master asserts the signal with START* and drives the address and transfer mode bits with the appropriate data.

(2) S1: All modules sample the address and transfer mode lines.

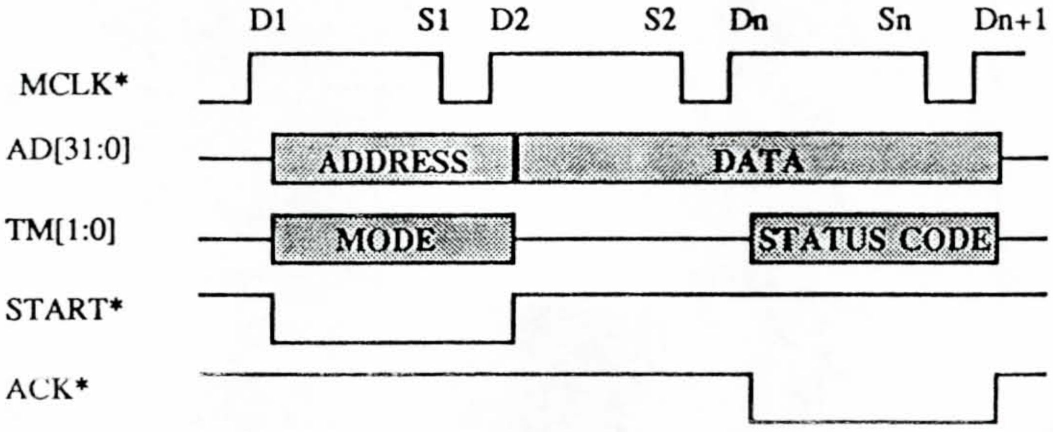
(3) D2: Master places data on the bus and waits for an acknowledge cycle.

(4) Dn: Addressed Slave encodes status information on the transfer mode bits (TM[1:0]*) and asserts the signal ACK*.

(5) Sn: Master samples the transfer mode lines and takes action based on status received from slave module.

(6) Dn+1 current or next bus Master must drive ACK* to a determinate state since the Slave stops driving the bus. Also, the Master stops driving the address/data lines.

Figure 4.2. Single-Word NeXTbus Write Transaction.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989) Reorder Product N#6010, pg. 4-10, Table 4-7.

The next type of transactions to be introduced are burst transfers. During burst transactions, multiple data words are transferred to and from sequential addresses. There are two words transferred per major clock cycle (one word every 40 ns). Only 32-bit words are supported during burst transfers. The user does not have the option of selecting a byte or halfword transaction as in single-word transfers. The size of the burst can be 4, 8, 16 or 32 words. The number of words to be transferred is encoded in the address lines bits 2 through 6 (AD[2:6]). Table 4.1 shows the burst size encoding scheme. The transaction type is still encoded as shown in Table 3.3 by using the two lsb's of the address lines and the two transfer mode bits.

During burst transfers, a mechanism called flow control is used to control the data rate on the bus. The NeXTbus supports both Master and Slave flow control. The Master uses the DRQ* signal to indicate it can sink two more words during a read transaction or that it can source two more words during a write transaction. The sink is the module receiving data during a transaction and the source is the module transmitting data. On the other hand, the Slave uses the signal TM0* to indicate it can source two words during a read transaction or that it can sink two words during a write transaction. The sink signal does not affect the current major clock cycle only subsequent cycles. An assumption at the beginning of every burst transaction is that every module must be able to sink two words. The sink signal is assumed asserted at the beginning of a burst transaction so two words are always transferred. Figure 4.3 shows a timing diagram for a four word burst read transaction. The sequence of events depicted on the timing diagram are as follows:

Table 4.1. Burst Size Encoding Scheme.

BURST STARTING					*BURST SIZE*	
AD6*	**AD5***	**AD4***	**AD3***	**AD2***	**WORDS**	**ADDRESS**
X	X	X	X	H	(illegal - 2 Words burst invalid)	
X	X	X	H	L	4	A[31:4]0000
X	X	H	L	L	8	A[31:5]00000
X	H	L	L	L	16	A[31:6]000000
H	L	L	L	L	32	A[31:7]0000000

SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg. 4-2, Table 4-2.

(1) D1: Master asserts the START*, TM[1:0]*, DRQ* signals and places the appropriate address on the bus. The signal DRQ* indicates to the Slave module that the Master can receive four words of data. The Master deasserts the ACK* and DSTB* signals.

(2) S1: Slave modules sample the address and transfer mode bit signal lines.

(3) D2: Master stops driving the address lines, ACK*, TM[1:0]* and START* signals. At this time, the Master is waiting for the signal TM0* to be asserted to start receiving data.

(4) Dn: Addressed Slave deasserts the TM1* and ACK* signals, places data on the bus and asserts the signal TM0*. Also, the DSTB* signal is asserted with the data.

(5) **Sn**: Master samples the TM0* and ACK* signals to verify transfer is still active.

(6) **Dn+1**: Slave places the last two words of the transfer on the bus along with the signal DSTB*. The Master deasserts the signal DRQ* which indicates to the slave that it can only sink two more data words.

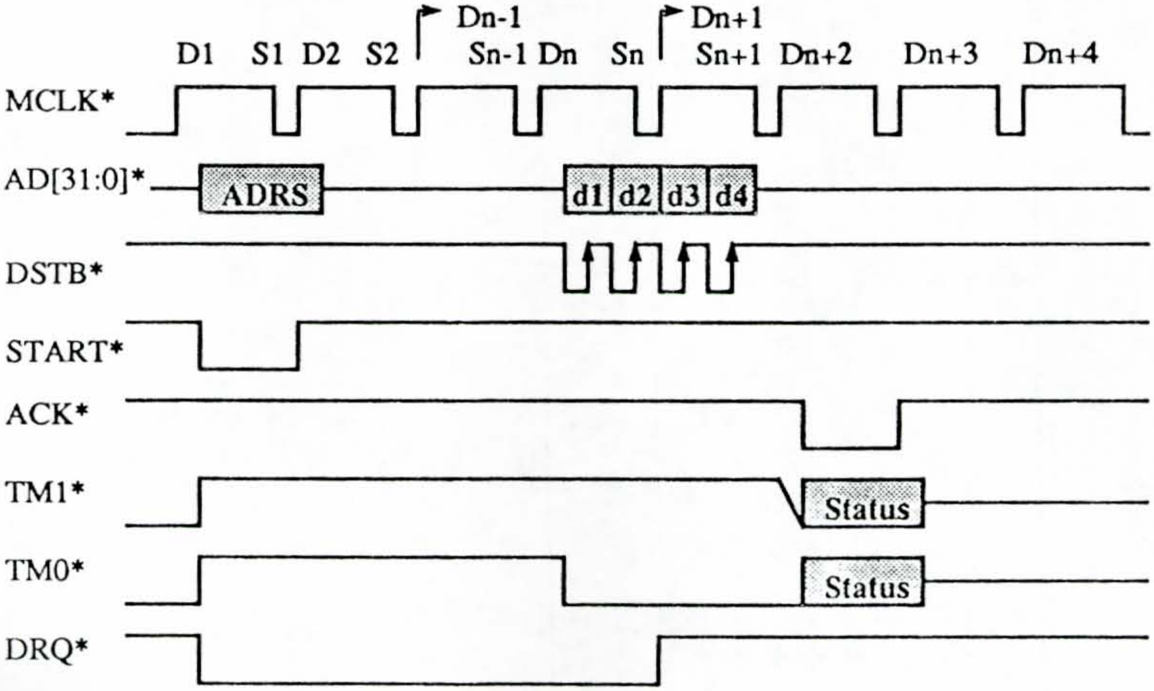
(7) **Sn+1**: The Master samples the transfer mode bits and ACK* signals to determine if the transaction is still active.

(8) **Dn+2:** Slave acknowledges the transaction by asserting the ACK* signal and encodes the status information on the two transfer mode bits.

(9) **Sn+2**: Master samples the ACK* and TM[1:0]* signals and takes appropriate action based on the status received from the Slave module.

(10) **Dn+3:** Slave stops driving the ACK*, DSTB*, AD[31:0]* and TM[1:0]* signals. Master stops driving the START* and DRQ* signal lines. At this time, the new bus owner shall drive the START*, ACK* and DSTB* signal lines to a determinate state.

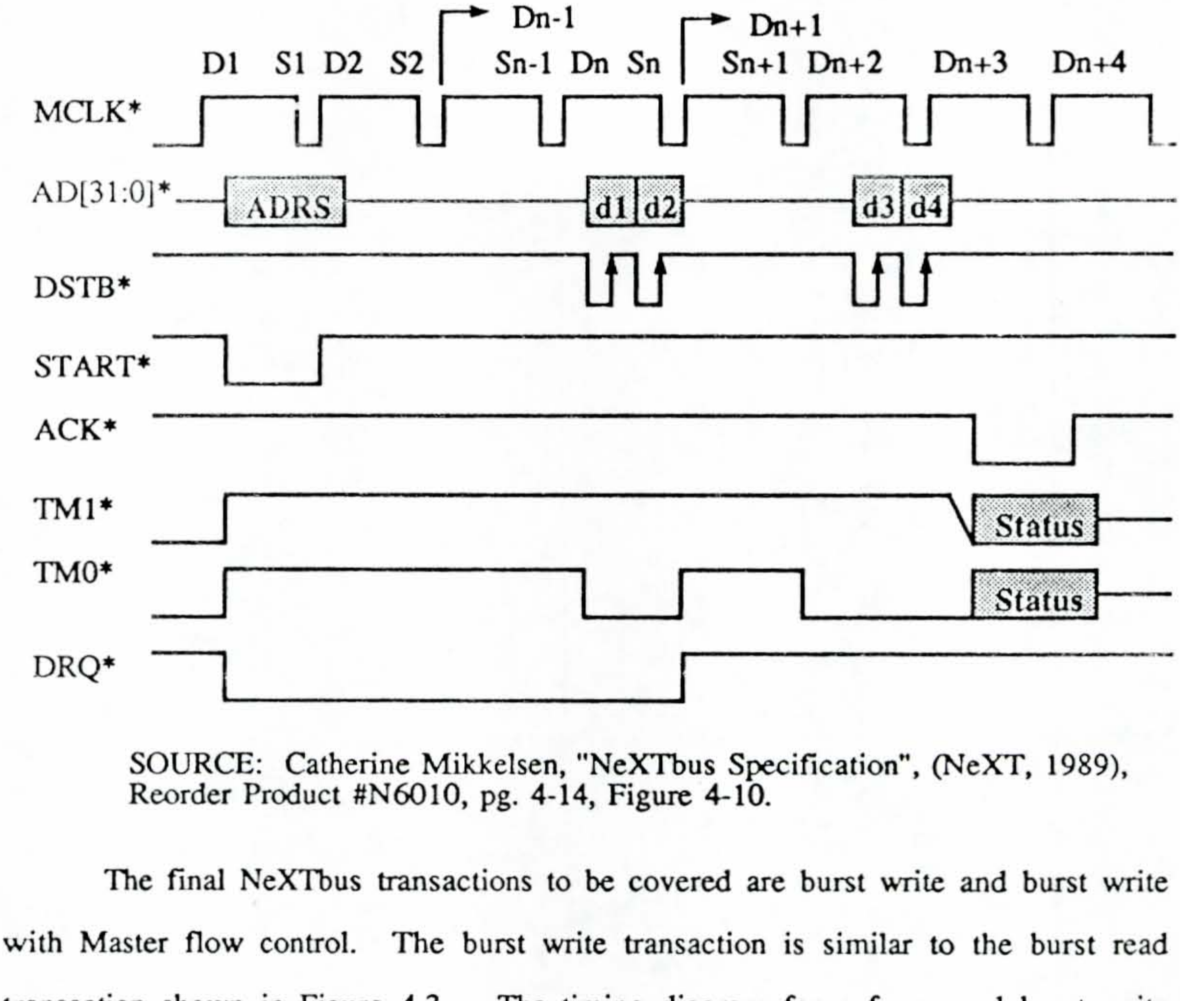
Figure 4.3. Timing Diagram for a Four Word Burst Read Transaction.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg. 4-13, Figure 4-9.

The next type of transaction to be introduced is a burst read with Slave flow control. The timing diagram for this transaction is shown in Figure 4.4. The difference is during cycle Dn+1 where the Slave module deasserts the signal TM0* indicating to the Master that no words shall be valid during the current major clock cycle. The Slave asserts the signal TM0* again during cycle Dn+2 and places the last two data words on the bus.

Figure 4.4. Timing Diagram for a Four Word Burst Read with Slave Flow Control.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg. 4-14, Figure 4-10.

The final NeXTbus transactions to be covered are burst write and burst write with Master flow control. The burst write transaction is similar to the burst read transaction shown in Figure 4.3. The timing diagram for a four word burst write transaction is shown in Figure 4.5. The sequence of events depicted on the timing diagram are as follows:

(1) D1: Master asserts the signal START*, places the address on the bus and encodes the type of transfer on the transfer mode bits. The signals DSTB* and ACK* are deasserted by the Master.

(2) S1: Slave modules sample the address and transfer mode bits.

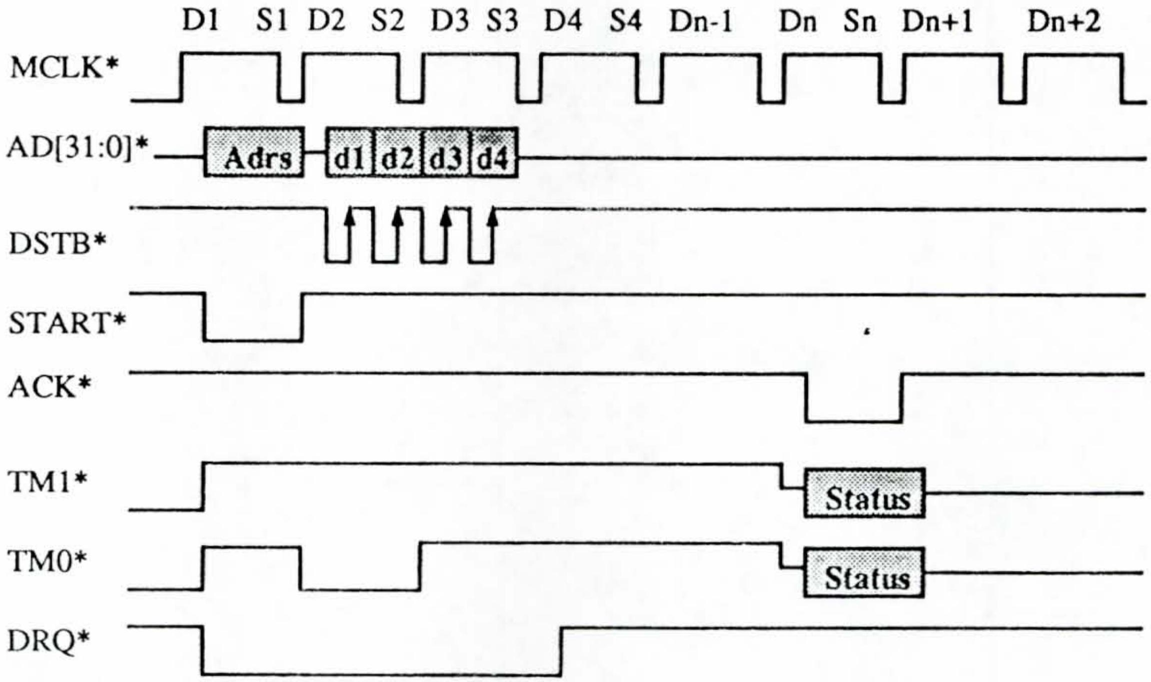
(3) D2: Master places data on the bus and asserts the DSTB* and DRQ* signals. The addressed Slave asserts the signal TM0* indicating to the master module that it can sink four words of data.

(4) S2: Master samples the ACK* signal to determine if transaction is still active.

(5) D3: Master places the last two words on the bus and asserts the signal DSTB*. The signal DRQ* is still asserted indicating to the Slave that two more data words are valid during this major clock cycle. The Slave deasserts the signal TM0* indicating that it can only sink two more data words.

(6) S3: Master samples the ACK* signal to determine if the transaction is still active.

Figure 4.5. Timing Diagram for a Four Word Burst Write Transaction.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg. 4-15, Figure 4-11.

(7) D4: Master stops driving the data bus, DSTB* and DRQ*signals.

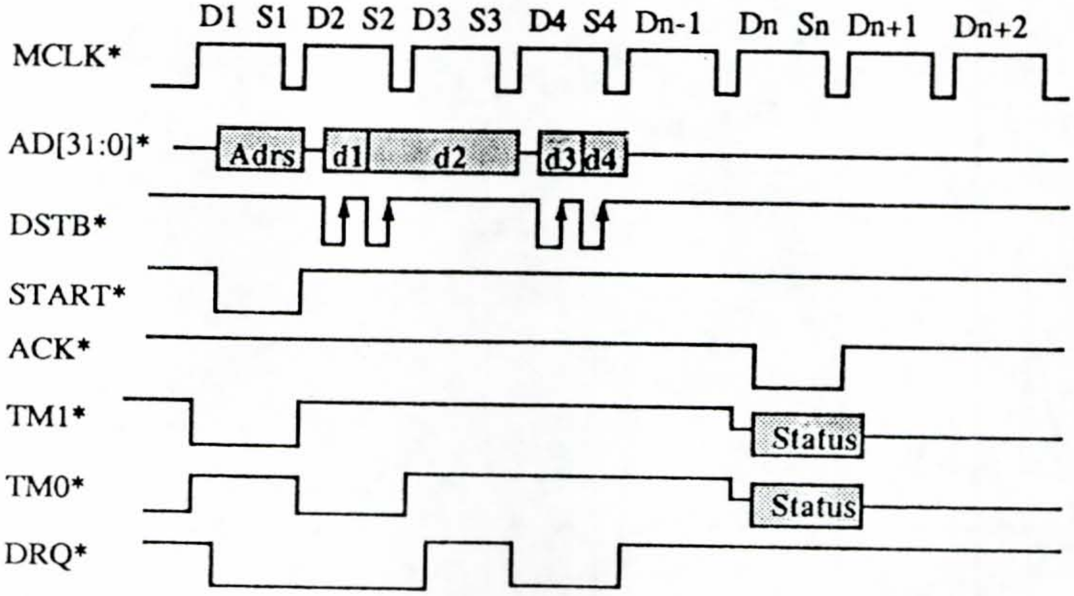
(8) Dn: Slave begins the acknowledge cycle by asserting the ACK* signal and encoding the status of the transaction on the two transfer mode bits.

(9) Sn: Master samples the ACK* and TM[1:0]* signals and takes appropriate action based on the status received.

(10) Dn+1: Slave module stops driving the two transfer mode bits and ACK* signal. The current owner or new bus owner must drive the START*, ACK* and DSTB* signal lines to a determinate state.

To conclude this chapter, Figure 4.6 shows a burst write transaction with Master flow control. The timing diagram is similar to Figure 4.5 except that the DRQ* signal is deasserted at edge D3. This indicates to the Slave module that two data words are not valid during the D3 major clock cycle. The DRQ* signal is asserted again at edge D4 and the two remaining data words are transferred. The Master module then waits for the Slave module to acknowledge the transfer.

Figure 4.6. Timing Diagram for a Four Word Burst Write with Master Flow Control.



SOURCE: Catherine Mikkelsen, "NeXTbus Specification", (NeXT, 1989), Reorder Product #N6010, pg. 4-16, Figure 4-12.

CHAPTER V

NEXTBUS INTERFACE CHIP OVERVIEW

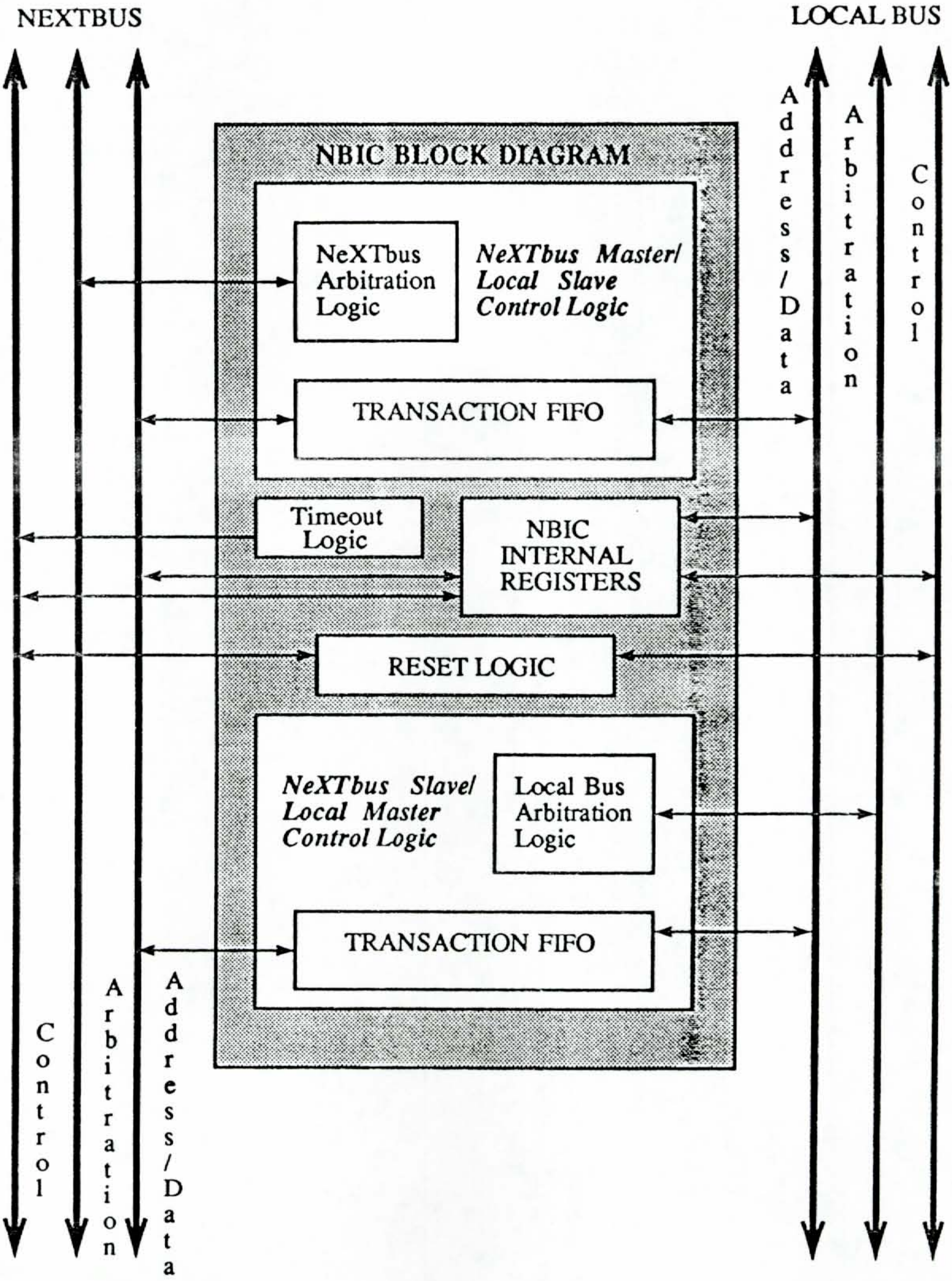
Introduction to the NBIC

The NBIC is a 144-pin CMOS VLSI chip which is used to simplify the logic required to interface any board to the NeXTbus. The NBIC was designed to interface specifically with the Motorola 68030 microprocessor. The NBIC resides between the NeXTbus and a local bus. In our case, the local bus shall communicate with an array of coprocessors. The chip contains arbitration logic which participates in arbitration contests on both buses and performs byte swapping. Byte swapping is performed to change between the Motorola 68030 Big-Endian byte ordering to a NeXTbus Little-Endian byte order. In our Host-to-Motherboard interface board, the NBIC shall not participate in NeXTbus arbitration because the board shall be a Slave only board. The NBIC is composed of five internal registers, NeXTbus Master/Local Slave control logic, NeXTbus Slave/Local Master control logic, timeout logic and reset logic. Figure 5.1 shows a top-level block diagram of the different components of the NBIC, which shall be covered in this chapter (Mikkelsen, "NeXTbus Interface Chip Specification", 1989).

NBIC Internal Registers

The five internal programmable registers in the NBIC are the NBIC ID register, control register, configuration register, interrupt register and the interrupt mask register. Each board's slot address space contains reserved address locations to define configuration and interrupt information about the board. The NBIC ID register is the only register out of the five that can physically be located in the board's memory

Figure 5.1 Top-level Block Diagram of NBIC.

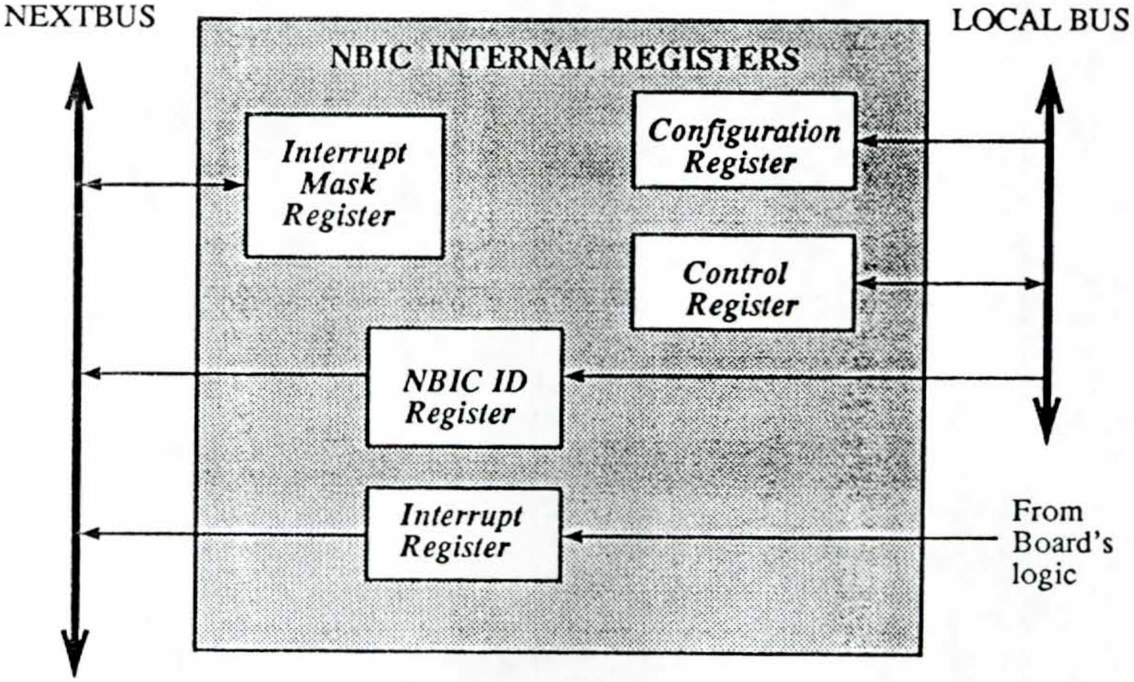


SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 2-2, Figure 2-1.

space outside the NBIC. The NBIC registers with the exception of the two interrupt registers can be read or written through the local bus. Figure 5.2 shows which registers are accessed by the local bus and which are accessed by the NeXTbus.

The NBIC ID register is used to store a 16-bit board identification number , a 15-bit manufacture identification number and a valid bit. Both of the identification values are used by the software to identify the type of board(s) inside of the NeXT cube. After power-up, a value needs to be written into the NBIC ID register and the VALID bit must be set. When the VALID bit is set, it indicates to bus Masters that the identification numbers have been written by the board. The contents of the NBIC ID register are written through the local bus and can be read by any bus Master on the NeXTbus. From the local bus, the register is written as one 32-bit word to address 4 hex. From the NeXTbus, the contents of the register are byte mapped across four 32-bit words. Figure 5.3 shows the address locations for the different bytes in the

Figure 5.2 Access to NBIC Internal Registers.

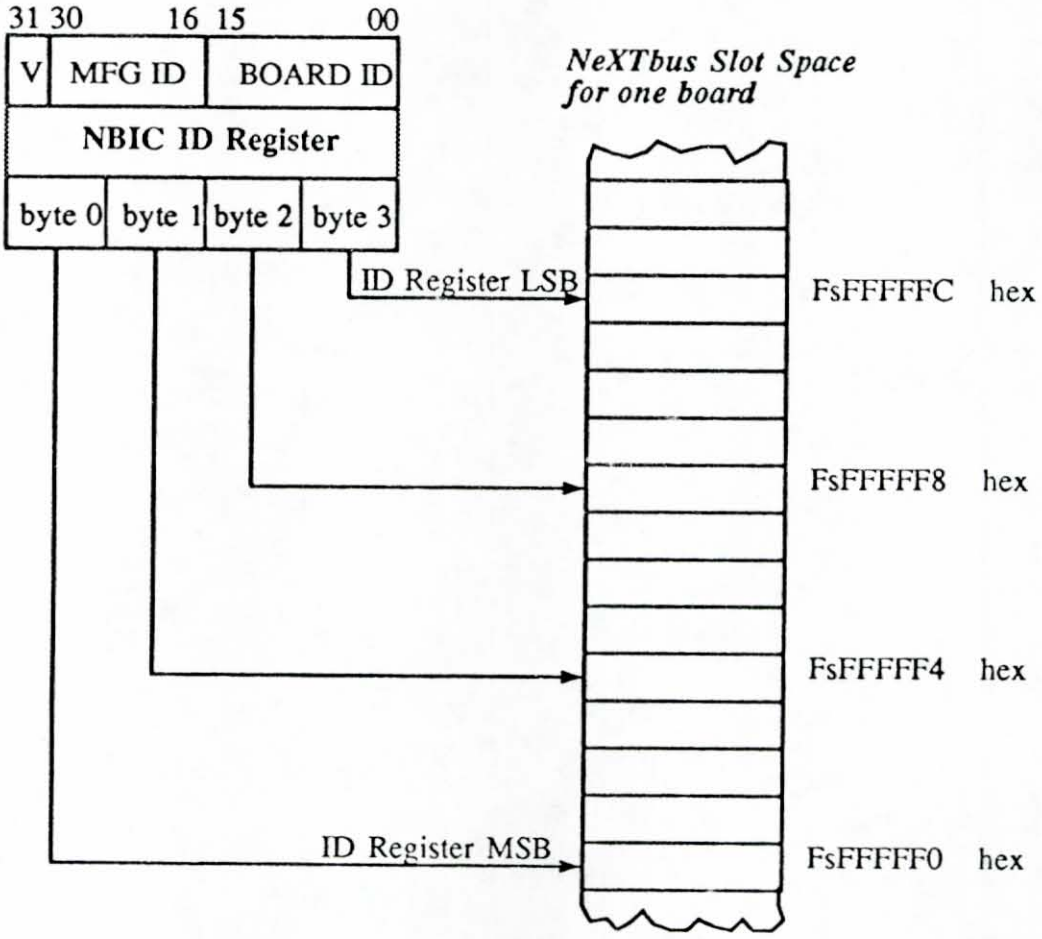


SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 4-2, Figure 4-1.

NeXTbus slot space. This addressing scheme is used since byte-wide devices such as Random Access Memory (RAM) might be used externally to the NBIC. Recall, that the NBIC ID register does not have to reside inside the NBIC address space.

The NBIC control register is used to control error conditions and operation of the chip. The contents of the register can be written or read through the local bus on the board. The control register is composed of two control bits denoted by STFWD and IGNSID0 and one error bit denoted by RMCOL. The store and forward word (STFWD) bit is used to enable this option during write operations. If this bit is set to one, then the NBIC immediately sends a termination signal to the local bus (if

Figure 5.3 NBIC ID Register in the NeXTbus Slot Space Address.



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 4-4. Figure 4-3.

there is room to store another data word). This allows for data words to be pipelined which speeds up transfers on the local bus. If this bit is set to zero, then the NBIC waits until the data is acknowledged by the NeXTbus Master before sending the final termination signal to the local bus. Write errors are not reported to the local bus when the store and forward option is selected. The ignore slot id 0 (IGNSID0) bit is used to control how much address space is used by the board. If this bit is set to one, then the address space of the slot n+1 is used; therefore, increasing the total address space of the board in slot n to 512 Megabytes (Mbytes) of memory. The final bit is the read-modify-write cycle collision (RMCOL) bit. This bit is set to one by the NBIC when it receives a retry acknowledge during a 68030 read-modify-cycle. Also, the NBIC issues a NeXTbus error when the RMCOL bit is set. Read-modify-cycles are part of the Motorola's 68030 microprocessor instruction set.

The value for the configuration register is set during the power-up sequence by using pullup resistors on the local bus. This register is configured during power-up and its contents can not be changed. The configuration register is composed of the slot id numbers, slave interrupt enable bit, external slot select bit, slot space decode bit and the external ID register enable bit. The slot id bits (31:28) define the location of the board in the NeXT cube. The value of the bits are defined in Table 5.1. The board designer(s) must install 10,000 Ohm resistors between the appropriate address lines (31:28) and the edge connectors of the board. The value of the slot id numbers shall be hard-wired on the backplane as defined in Table 5.1.

The slave interrupt enable bit (SINTEN) controls the operation of the GLAVE*/SINT* signal. If the bit is set to one, then the GSLAVE*/SINT* signal is configured as an input to the NBIC. When the signal SINT* is asserted on the local bus, the NeXTbus signal GINT* is asserted. The signal GINT* is an interrupt signal on the NeXTbus which propagates to all boards on the backplane. The signal SINT* is generated by the board's logic to interrupt the NeXTbus Master. If the slave inter-

Table 5.1. Slot ID Numbers per Slot.

Configuration Register Bits	(AD31) SID3	(AD30) SID2	(AD29) SID1	(AD28) SID0
SLOT 0	GND	GND	GND	GND
SLOT 2	GND	GND	+5V	GND
SLOT 4	GND	+5V	GND	GND
SLOT 6	GND	+5V	+5V	GND

SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 4-6.

rupt enable bit is set to zero, then the signal GLAVE* is used as an output. The signal GSLAVE* shall be asserted by the NBIC when the board is a NeXTbus slave. The GSLAVE* and SINT* signals can not be used at the same time. The bit SINT-EN determines which of the two signals is used by the board's logic.

The local bus grant/external select (LBG/EXSEL) bit is also used to determine which of the two signals shall be used by the board's logic. If this bit is set to one, then the signal EXSEL is asserted by the NBIC when a NeXTbus Master references the slot address space or ID register (Fs000000 hex to FsFFFFF1 hex). This bit is used when the NBIC ID register is contained in the board's memory space versus the NBIC. If the bit is set to zero, then the local bus grant (LBG) signal shall be used by the board's logic (This signal is used when there are more than two local bus masters).

The slot space decode bit (SSDECODE) is used to enable or disable the board's slot space address. If this bit is set to one, then addresses in the range Fs000000 hex to FsFFFFE4 hex are passed to the local bus. If this bit set to zero, then any references which are made to the board slot space address (does not include the NBIC ID register or Interrupt registers) results in a timeout error. Finally, the external ID register enable bit (EXIDREGEN) is used to enable or disable the internal NBIC ID register. If this bit is set to one, then the NBIC ID register is disabled and all references between FsFFFFF0 hex through FsFFFFFC hex are passed to the local bus. If this bit is set to zero, then the NBIC ID register is located internal in the

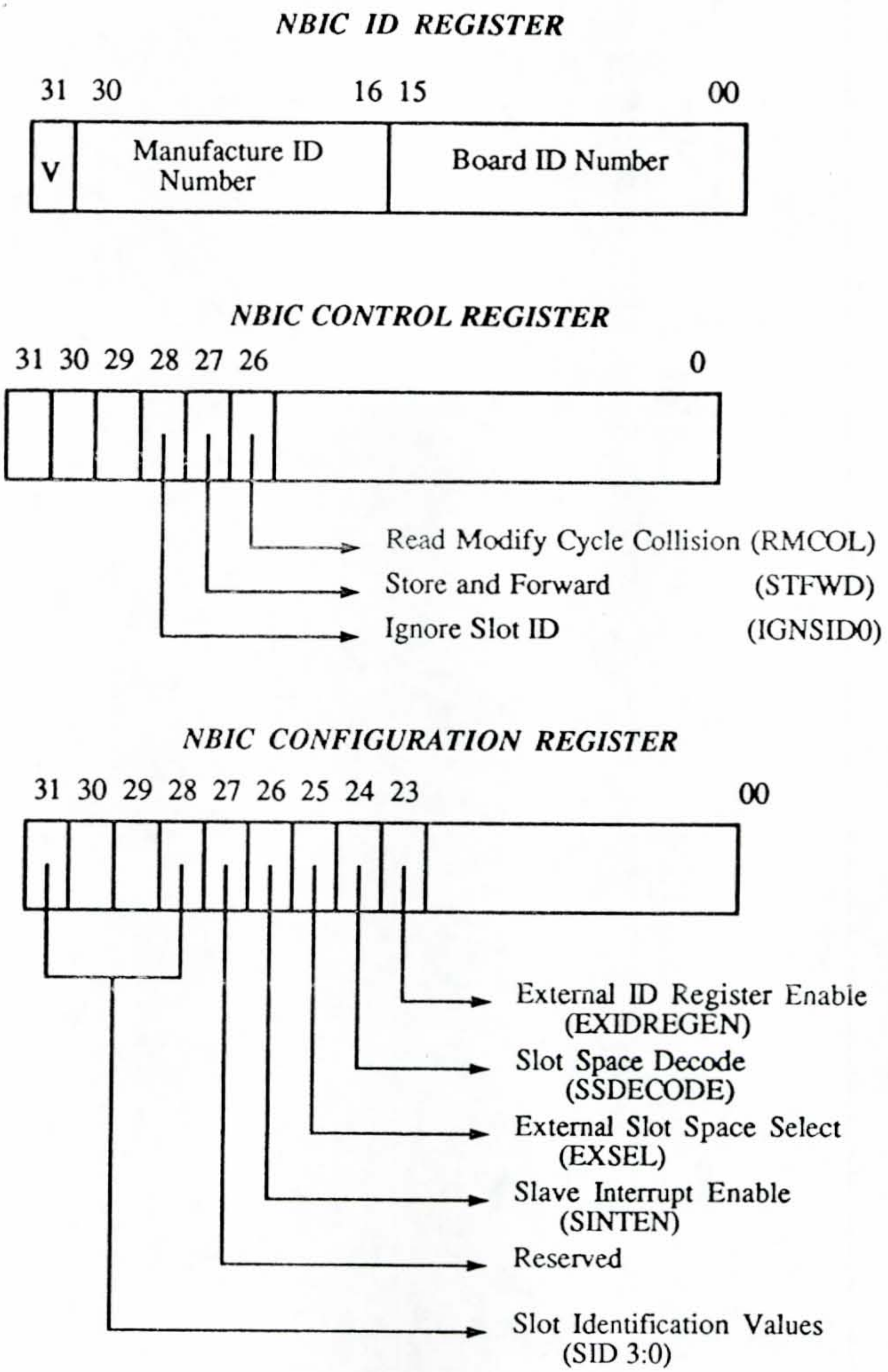
NBIC. When the NBIC ID register is located externally, the SSDECODE and EXIDREGEN bits must both be set to one.

The two interrupt registers are used to generated an interrupt signal (GINT*) on the NeXTbus. The interrupt register is located at address FsFFFFE8 hex and the interrupt mask register is located at address FsFFFFFEC hex. The interrupt register is a Read only register and the interrupt mask register is a Read/Write register. If the interrupt mask register is set to one and the board's logic asserts the signal SINT*, then the NeXTbus interrupt signal (GINT*) shall be asserted. If the mask interrupt register is set to zero, then an interrupt is not generated on the NeXTbus. Figure 5.4 shows the format for all the NBIC internal registers except the two interrupt registers. The two interrupt registers are one-bit registers with the GAD7* address/data line being the appropriate bit to either read or write.

NeXTbus Master/Local Slave Control Logic

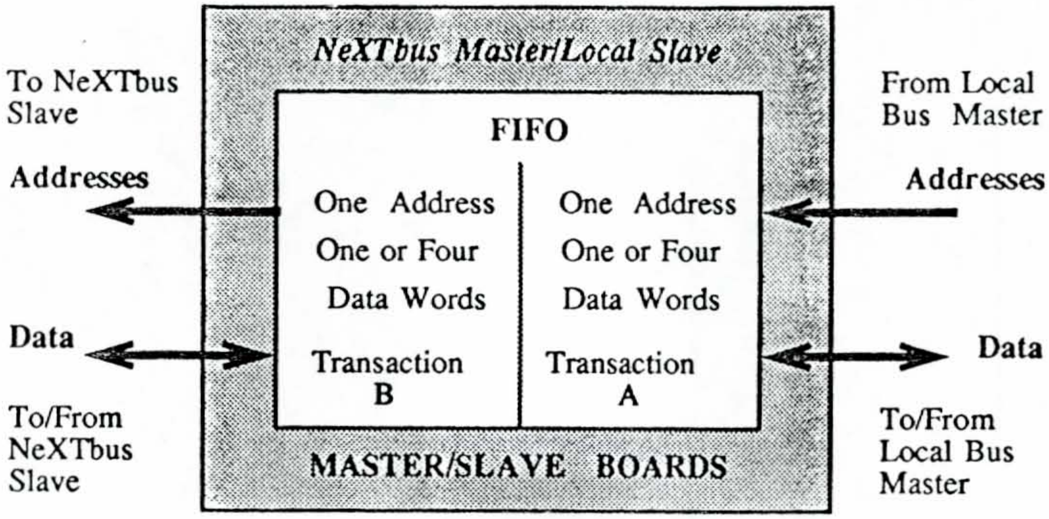
The NeXTbus Master/Local Slave control logic is used during all transactions when the NBIC is a slave to a local bus master. This control logic is used by master/slave boards. A slave only board does not used this control logic on the NBIC. This control logic contains a NeXTbus arbiter and a transaction FIFO. The arbitration logic participates in arbitration cycles to obtain ownership of the NeXTbus. The FIFO is used to buffer address and data information for a maximum of two transactions. The FIFO serves a dual purpose. It performs speed matching between a fixed-rate NeXTbus and a variable rate local bus and it allows transactions two be pipelined. The FIFO has the capability to store two transactions back to back. A transaction is composed of one address and either one word for single-word transfers or four words for burst transfers. NBIC only supports a burst size of four 32-bit words during burst transfers. Figure 5.5 shows the transaction FIFO for the NeXTbus Master/Local Slave control logic.

Figure 5.4. Format for Internal NBIC Registers.


NBIC ID REGISTER
31
30
16
15
00
V
Manufacture ID Number
Board ID Number
NBIC CONTROL REGISTER
31 30 29 28 27 26
0
Read Modify Cycle Collision (RMCOL)
Store and Forward (STFWD)
Ignore Slot ID (IGNSID0)
NBIC CONFIGURATION REGISTER
31 30 29 28 27 26 25 24 23
00
External ID Register Enable (EXIDREGEN)
Slot Space Decode (SSDECODE)
External Slot Space Select (EXSEL)
Slave Interrupt Enable (SINTEN)
Reserved
Slot Identification Values (SID 3:0)

SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specifications", (NeXT, 1989), pg. 4.3-4.5, Figures 4.2,4.4,4.5.

Figure 5.5. NeXTbus Master/Local Slave Transaction FIFO.



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 2-3, Figure 2-2.

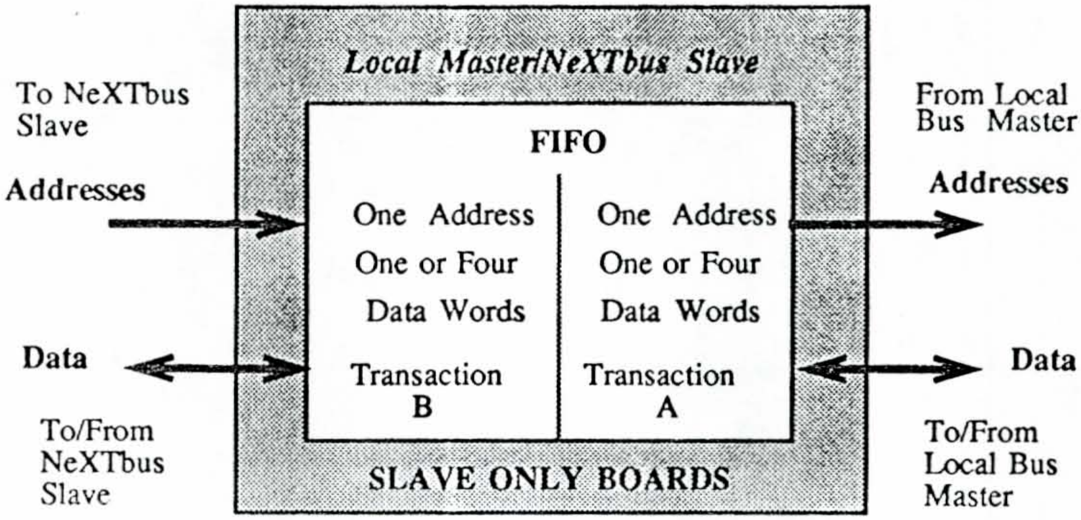
Local Master/NeXTbus Slave Control Logic

This control logic is used when the NBIC is a NeXTbus slave and a local bus master. On the NeXT-to-Motherboard interface board to be designed, this is the only transaction control logic which shall be used since the board is a slave only board (does not participate in NeXTbus arbitration cycles). This transaction control logic contains a local bus arbiter and a FIFO. The arbitration logic is used to obtain ownership of the local bus and the FIFO is used to buffer the address and data. The FIFO serves the same purposes as explained in the NeXTbus Master/Local Slave control logic section. Figure 5.6 shows the transaction FIFO for the Local Master/NeXTbus Slave control logic.

Timeout Logic

All transactions on the NeXTbus must be completed by 255 MCLK cycles which is approximately 20.4 microseconds (us). The NBIC timeout control logic consist of an R-S flip-flop and a 8-bit counter. The flip-flop is set by the START* NeXTbus signal (NBIC signal GSTART*) and the counter is enabled. The ac-

Figure 5.6. Local Master/NeXTbus Slave Transaction FIFO.



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 2-5, Figure 2.4.

knowledge signal ACK* (NBIC signal GACK*) is used to clear the flip-flop and the counter. If the counter reaches to a count of 255, then the timeout logic issues a bus error and the signal GACK* is asserted at cycle 256. The CPU board also asserts the ACK* signal during cycle 256 in case there is no board in the addressed slot.

Reset Logic

The reset logic contains a Schmitt trigger buffer input for a reset signal from either the NeXTbus or the local bus. When the reset signal on the NeXTbus is asserted, it triggers a pulse generator circuit which shall assert the reset signal on the local bus for 1.28 us. The same condition exits if the reset signal on the local bus is asserted. The pulse generators in the reset logic triggers when a negative edge is detected.

CHAPTER VI

NBIC SIGNAL DESCRIPTION

The purpose of this chapter is to introduce all the signals of the NBIC. The signals are grouped into six different groups based on the function they perform. The six classes are address/data, arbitration, control, utility, clocks and power. All of the NeXTbus signals are prefaced with a "G", which stands for global, and shall not be covered in this chapter since they are described in Chapter III. There are only four signals which are not defined in Chapter III and shall be defined in this chapter along with all the NBIC local bus signals. Table 6.1 shows all the global and local bus signals of the NBIC. All the information for this chapter was obtained from the "NeXTbus Interface Chip Preliminary Specifications" by Mikkelsen.

The three clock signals BUSCLKIN, GBUSCLKO and GMCLKSELO are only used in the CPU board in slot 0. BUSCLKIN is the 25 MHz input clock on the CPU board which is used to generate GBUSCLKO and GMCLKSELO. These two output clocks from the NBIC in the CPU board are then used by all boards on the backplane. The last signal which is not defined in Chapter III is GDSTBO*. The NBIC has two data strobe signals denoted by GDSTB* and GDSTBO*. The signal GDSTB* is an input to the NBIC and the signal GDSTBO* is an output of the NBIC. However, the NeXTbus only has one data strobe signal on its backplane. The signal GDSTBO* from the NBIC of the NeXT CPU board (Slot 0) should be connected to the GDSTB* input signal of the NBIC located in the NeXT-to-Motherboard interface board. This information should be confirmed with a technical representative from NeXT Corporation. The preliminary specifications does not state the intent of the GDSTBO* signal.

Table 6.1 NBIC NeXTbus and Local Bus Signals.

SIGNAL	DESCRIPTION	TYPE	# PINS
GLOBAL SIGNALS			
GAD[31:0]*	Global Address/Data	I/O	32
GRQST*	Global Request	I/O	1
GSTART*	Global Start	I/O	1
GDRQ*	Global Data Request	I/O	1
GACK*	Global Acknowledge	I/O	1
GTM[1:0]*	Global Transfer Mode	I/O	2
GDSTB*	Global Data Strobe	I	1
GDSTB0*	Global Data Strobe Out	O	1
GSP[3:0]*	Global System Parity	I/O	4
GSPV*	Global System Parity Valid	I/O	1
GRESET*	Global Reset	I/O	1
GINT*	Global Interrupt	O	1
GARB[3:0]*	Arbitration Number	I/O	4
BUSCLKIN	25 MHz Clock	I	1
GBUSCLK	Global Bus Clock	I	1
GBUSCLK0	Global Bus Clock Out	O	1
GMCLKSEL*	Major Clock Select	I	1
GMCLKSEL0*	Major Clock Select Out	O	1
PON	Power On	I	1
LOCAL SIGNALS			
LAD[31:0]*	Local Address/Data	I/O	32
BR*	Bus Request	O	1
BG*	Bus Grant	I	1
BGACK*	Bus Grant Acknowledge	O	1
LBR*	Local Bus Request	I	1
LGB/EXSEL	Local Bus Grant/External Select	O	1
NCS*	NBIC Chip Select	I	1
GBCYC*	Global Bus Cycle	I	1
GSLAVE*/SINT*	NeXTbus Slave/NeXTbus Interrupt	I/O	1
GMASTER*	Global Master	O	1
FB*	Function Bit	O	1
SIZ[1:0]	Transfer Size	I/O	2
DS*	Data Strobe	I/O	1
AS*	Address Strobe	O	1
ASIN*	Address Strobe In	I	1
R/W*	Read/Write	I/O	1
RMC*	Read Modify Cycle	I/O	1
DSACK[1:0]	Data Size Acknowledge	I/O	2
BREQ*	Burst Request	I/O	1
BACK*	Burst Acknowledge	I/O	1
STERM*	Synchronous Termination	I/O	1
HALT*	Halt	I/O	1
BERR*	Bus Error	I/O	1
LRESET*	Local Bus Reset	I/O	1
OE	Output Enable for Test	I	1
CPUCLK	CPU Clock	I	1
LCLK	Local Bus Clock	I	1

SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 3-2, Table 3-1.

The NBIC local bus signals are used to interface the NBIC to the board's logic. A brief description of all the NBIC local bus signals are defined below:

Local Address/Data (LAD [31:0])

These are the 32 bidirectional multiplexed lines which carry the address at the beginning of a transaction and data information later in the transaction.

Bus Request (BR)*

This is an open-drain wired-"OR" signal which is asserted by the NBIC or any other local bus master that wants to obtain ownership of the local bus. An external arbiter is used to grant ownership of the bus.

Bus Grant (BG)*

This signal is an input to the NBIC and is asserted by the external arbiter when the NBIC has won ownership of the local bus. This signal remains asserted by the external arbiter until the NBIC deasserts the signal BR*.

Bus Grant Acknowledge (BGACK)*

This is an open-drain signal which is asserted by the NBIC or any other local bus master when it obtains ownership of the local bus. This signal is deasserted by the bus owner after the transaction is complete. The local bus can not be accessed by any other module while this signal is asserted.

Local Bus Request (LBR)*

This signal is only used when there are three or more potential local bus masters. In a Slave only board, the NBIC is always the master of the local bus and no local bus arbitration takes place. This signal shall be pulled up by 100,000 Ohm resistor on the NeXT-to-Motherboard interface board.

Local Bus Grant/External Select (LBG/EXSEL)

If the LBG signal is selected in the configuration register, then this signal is asserted by the NBIC when the internal local bus arbiter grants bus ownership to another bus master. This signal is not used in a Slave only board. If the EXSEL signal is selected, then the signal is asserted by the NBIC when reference is made to the boards slot address space or ID register.

NBIC Chip Select (NCS)*

This signal is asserted by the board's logic when a transaction with the internal NBIC registers is to take place.

Global Bus Cycle (GBCYC)*

This signal is asserted by the board's logic when a local bus master is to perform a transaction on the NeXTbus. This signal is not used on a Slave only board.

Global Master (GMASTER)*

This signal is asserted by the NBIC when it is in the process of arbitrating for the NeXTbus or is the NeXTbus Master.

Global Slave (GSLAVE/SINT*)*

If the GSLAVE* signal is selected in the configuration register, then the signal is asserted when the NBIC is a NeXTbus Slave. If the signal SINT* is selected, then the board's logic can assert this signal to generate a NeXTbus interrupt (GINT*). The interrupt mask register bit (GAD7*) must be set to one for the interrupt to be generated on the NeXTbus.

Function Bit (FB)*

There is not set protocol by NeXT on this signal. This bit is intended to be used with the Motorola's 68030 function codes. This bit is asserted by the NBIC when it is the master of the local bus.

Transfer Size (SIZ[1:0])

These are tri-state signals which define the number of bits transferred on the data bus during local bus transactions. These signal lines are asserted by the local bus master. Table 6.2 shows the encoding of these signals.

Table 6.2. Encoding of the Transfer Size Signal Lines.

BITS	SIZ1	SIZ0
32	L	L
24	H	H
16	H	L
8	L	H

SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 3-8, No Table number assigned.

Data Strobe (DS)*

This signal is asserted by the local bus master during transactions. During a read transaction, the signal DS* is asserted to indicate to the external device to place data on the local bus. During a write transaction, the signal DS* is asserted when the data is placed on the local bus.

Address Strobe (AS)*

This signal is asserted when the NBIC is the local bus master indicating to the board's logic that there is a valid address on the local data bus. This signal is asserted throughout the duration of the transaction.

Address Strobe In (ASIN)*

This signal is asserted by the local bus master during any transaction with the NeXT-bus. The NBIC latches an address when the local bus master asserts the signal ASIN*. This signal must be asserted throughout the duration of the transaction.

Read/Write (R/W)*

This is asserted by the local bus master to indicate the type of transfer to be performed. If the a read transaction is to be performed, then this signal is asserted high. If a write transaction is to be performed, then this signal is asserted low.

Read Modify Cycle (RMC)*

This signal is asserted by the local bus master when a read-modify-cycle is in progress.

Burst Request (BREQ)*

This signal is asserted by the local bus master when a burst read or write transaction is to take place on the local bus. The burst transfer is then performed on the NeXT-bus. Recall that all NBIC burst transfers are four words. The NBIC does not support burst transfers of 8, 16 or 32 words.

Burst Acknowledge (BACK)*

This signal is asserted by the local bus slave to inform the local bus master that the local slave supports burst transactions. If this signal is not asserted, then the local bus slave does not support burst transactions.

Data Transfer and Size Acknowledge (DSACK[1:0])*

These signals are asserted by the local bus slave to terminate a transaction.

Synchronous Termination (STERM)*

This signal is asserted by the local bus slave during all local bus transactions. During a write transaction, the local bus slave shall assert the signal STERM* to inform the local bus master that it shall latch the 32-bit data word on the next trailing edge of LCLK. During a read transaction, the local bus slave shall assert the signal STERM* to inform the local bus master that a 32-bit data word shall be valid on the next trailing edge of LCLK. The STERM* or DSACK[1:0] signals are used to indicate termination of a transaction. Both are never used simultaneously to terminate transactions.

HALT (HALT)*

This signal is asserted by the local bus master to indicate the type of termination. If this signal is asserted along with the BERR* signal , then the current transaction on the local bus must be halted.

Bus Error (BERR)*

This signal is asserted by the local bus master if a timeout or error occurs on the local bus. If this signal is asserted along with the HALT* signal, then the current transaction must be halted. If a local bus slave asserts this signal while the NBIC is a local bus master, then an error termination shall be transmitted on the NeXTbus.

Local Reset (LRESET)*

This is the local reset signal. If asserted by the board's logic, then the NeXTbus reset signal (GRESET*) shall be asserted. This signal is asserted by the NBIC when the GRESET* signal is asserted on the NeXTbus. When the signal PON is deasserted, both of the reset signals are asserted. This signal shall be asserted by the NBIC during the power-up sequence.

Output Enable (OE)

When this signal is deasserted by the board's logic, every NBIC output is tri-stated. This signal can be used to isolate the NBIC during testing of the board.

Local Bus Clock (LCLK)

This clock is used by the NBIC when it is the local bus master. This clock is supplied by the board's logic and can run up to 16.66 MHz. In Slave only boards, the board's logic only generates this clock. This clock is selected by the NBIC when the signal BGACK* is asserted.

CPU Clock (CPUCLK)

This clock runs at 25 MHz and is selected when the signal BGACK* is deasserted.

CHAPTER VII

NBIC LOCAL BUS SPECIFICATIONS

Introduction to the NBIC Local Bus Interface

The NBIC was designed to interface primarily with the Motorola 68030 microprocessor. The 68030 uses a Big-Endian byte ordering scheme for its data bus. In Big-Endian buses, byte 0 is located in the most significant bits (msb's) of the data bus (31:24) and byte 3 is located in the least significant bits (lsb's) of the data bus (7:0). The NeXTbus is a Little-Endian bus which is the opposite of a Big-Endian bus. Byte 0 in a Little-Endian bus is in the lsb's of the data bus and byte 3 is in the msb's. Since the NBIC was designed to interface with the 68030, byte swapping is performed to directly support transfers with the NeXT CPU board. The local bus of the NBIC is a Big-Endian bus so that it can interface directly with a 68030. Big-Endian boards such as the CPU board in slot 0 have no problem communicating with each other. The problem is when a Little-Endian board tries to communicate with a Big-Endian board. There needs to be a way to detect word transfers between these two different byte ordering schemes and perform the required byte ordering. If bytes are only to be transferred between the same two byte ordering schemes, then byte swapping is not required. The purpose of this chapter is to introduce the different type of transactions supported by the NBIC on its local side. Only transactions where the NBIC is a local bus master shall be covered in this chapter since the NeXT-to-Motherboard interface board is a Slave only board. Also, the local bus on this board shall be a Big-Endian bus so that communication can take place with the 68030 based CPU board. The information for this chapter was obtained from the NeXTbus Interface Chip Preliminary Specification" by Mikkelsen.

Introduction to Local Bus Transactions

A local bus transaction is composed of three phases. The three phases are bus synchronization, actual transfer of data and bus release. The bus synchronization phase, which takes two and a half clocks, allows the NBIC and the bus to adjust to the different data rates of the NeXTbus and the local bus. The bus release phase accommodates the delay time for the different bus masters. This phase takes three clocks and it guarantees that the NBIC maintains bus ownership through multiple store and forward write transactions.

When the NBIC is a local bus master, it initiates all transactions and then waits for termination signals from the local bus slave. The transaction types for the NBIC as a local bus master are encoded in the size bits (SIZ[1:0]) and the two lsb's of the address/data lines. Table 7.1 shows the valid encoding schemes supported by the NBIC as a local bus master. The transactions shown in the table are valid during read and write transfers. The local bus slave performs transaction termination by encoding the status of the transfer on the STERM* or DSACK[1:0]*, HALT* and BERR* signals. These signals are encoded to indicate the completion and status of a transaction. The encoding schemes for the different valid transaction terminations are

Table 7.1. Valid Transaction Types When NBIC is Local Bus Master.

SIZ1	SIZ0	LA1	LA0	TYPE	VALID BITS
L	H	H	H	Byte 3	Bits [7:0]
L	H	H	L	Byte 2	Bits [15:8]
L	H	L	H	Byte 1	Bits [23:16]
L	H	L	L	Byte 0	Bits [31:24]
H	L	H	L	Halfword 1	Bits [15:0]
H	L	L	L	Halfword 0	Bits [31:16]
L	L	L	L	Word	Bits [31:0]
L	L	L	L	Burst	N/A

SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 2-15, Table 2-3.

shown in Table 7.2. The signal STERM* is a synchronous signal and must meet all timing parameters such as setup and hold times. The other signals in Table 7.2 are either synchronous or asynchronous. If they meet all the timing parameters, then they are treated as synchronous signals. If they do not meet the synchronous timing parameters, then they are treated as asynchronous and must be stable for one full cycle before the data is sampled.

Table 7.2. Encoding Schemes for Transaction Terminations.

STERM*	DSACK*0	DSACK*1	BERR*	HALT*	RESULT
H	H	H	H	H	Insert Wait
L	H	H	H	H	Sync 32-bit ACK
L	H	H	L	L	Sync Bus Error
L	H	H	L	L	Sync Retry
H	H	H	L	L	Async Bus Error
H	H	H	L	L	Async Retry
H	L	L	H	H	Async 32-bit ACK
H	L	H	H	H	Async 16-bit ACK
H	H	L	H	H	Aysnc 8 -bit ACK

SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 2-16, Table 2-4.

NBIC Internal Register Transactions

This section shall cover the transactions with the internal registers in the NBIC. Recall, that the control register is the only register which can be written and read from the local bus. The NBIC ID register can only be written from the local bus. A timing diagram showing an internal write to an NBIC internal register is shown in Figure 7.1. The sequence of events depicted on the timing diagram are described below:

Clock 1: The local bus master (board's logic) places the appropriate address on the bus (NBIC ID address is 4 hex and control register address is 0 hex) during the first phase of the clock. During the second phase, the local bus master asserts the signal ASIN* which is used to latch the address into the NBIC.

Clock 2: The address goes invalid at the beginning of the clock. During the second phase, the local bus masters places size information SIZ[1:0] on the bus, drives the signal R/W* low and asserts the signal NCS* to indicate to the NBIC that it is performing an internal register access cycle.

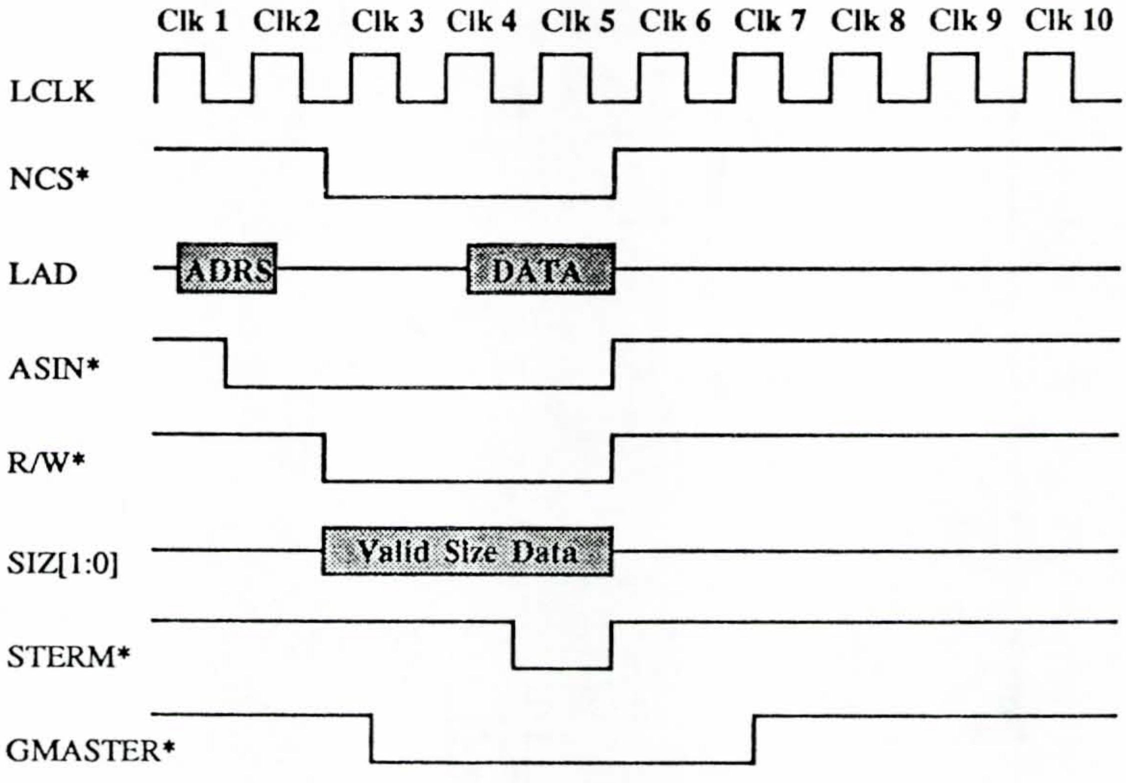
Clock 3: The NBIC asserts the signal GMASTER* during the first phase of the clock indicating that it is a local bus slave.

Clock 4: The local bus master places the appropriate data during the first phase of the clock. During the second phase, the local bus master asserts the signal STERM* indicating to the NBIC to latch data on the next trailing edge of LCLK.

Clock 5: The NBIC latches the data on the trailing edge of the clock and the local bus master deasserts the signals NCS*, STERM*, R/W*, SIZ[1:0] and ASIN*.

Clock 7: The NBIC deasserts the signal GMASTER* during the first phase of the clock.

Figure 7.1. NBIC Internal Register Write from Local Bus



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 6-20, Figure 6-13.

A read to the NBIC internal registers is only valid for the control register. Figure 7.2 shows the state timing diagram for an NBIC internal register read. The sequence of events depicted in the timing diagram are described below:

Clock 1: The local bus master places the address on the bus during the first phase of the clock. The signal ASIN* is asserted during the second phase of the clock so the NBIC can latch the address

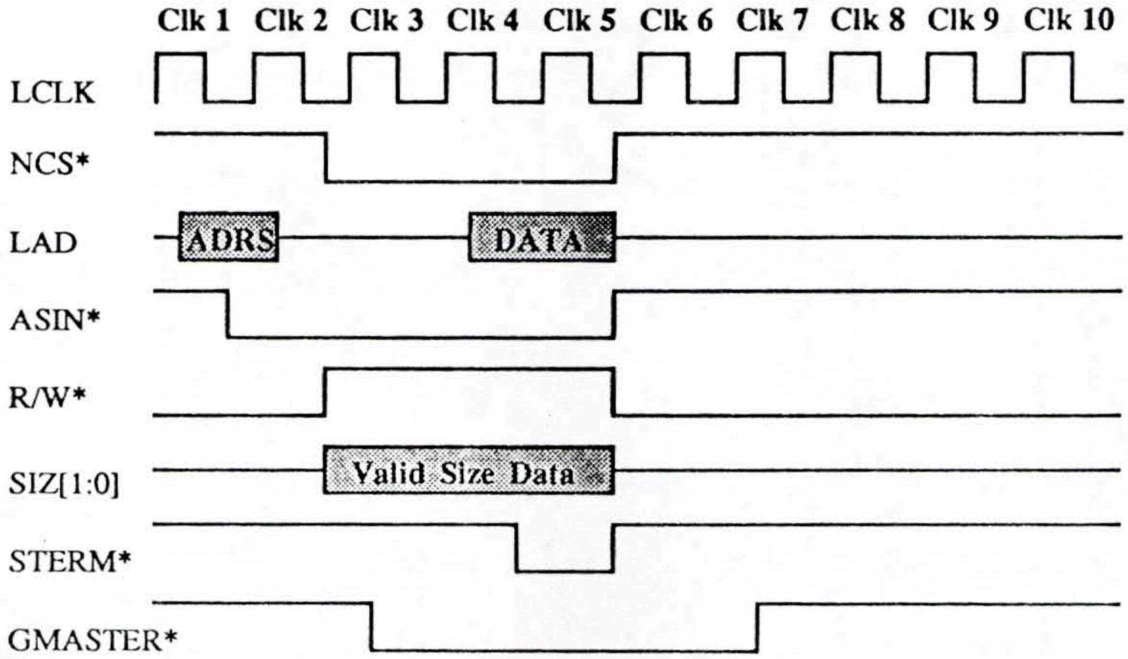
Clock 2: The address is no longer valid during the first phase of the clock. The local bus masters asserts the size information bits (SIZ[1:0]), drives the signal R/W* high and asserts the signal NCS*.

Clock 3: The NBIC asserts the signal GMASTER* indicating that it is a local bus slave.

Clock 4: The NBIC places the data requested during the first phase of the clock. The data is valid during the second phase of the clock and the NBIC asserts the signal STERM*.

Clock 5: The local bus masters latches the data on the trailing edge and deasserts the signals NCS*, R/W*, SIZ[1:0] and ASIN*. The NBIC deasserts the signal STERM*.

Figure 7.2. NBIC Internal Register Read from Local Bus.



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 6-19, Figure 6-12.

Clock 6: The NBIC stops driving the data bus.

Clock 7: The NBIC deasserts the signal GMASTER* during the first phase of the clock.

Single-Word Local Bus Transactions

This section covers single-word transactions which are initiated by the NBIC when it is a local bus master. The two transactions which shall be covered are single-word read and write. Figure 7.3 shows a timing diagram for a single-word write by the local master NBIC. The sequence of events depicted in the timing diagram are described below:

Clock 1: NBIC is performing bus synchronization during the first phase of the clock. During the second phase of the clock, the NBIC asserts the signals BGACK*, FB*, GSLAVE*, places and address on the bus, drives the size information bits (SIZ[1:0]) and the signal R/W* (drives R/w* low to indicate a write cycle).

Clock 2: The values on the address/data, R/W* and SIZ[1:0] signal lines are valid during the first phase of the clock. The NBIC asserts the signal AS* during the second phase of the clock so the local bus slave can latch the address. The signal AS* is valid throughout the transaction.

Clock 3: The address/data lines are invalid during the first phase of the clock. The NBIC places data on the bus and asserts the signal DS* so the local bus slave can latch the data.

Clock 4: The NBIC waits till the local bus slave asserts either the DSACK[1:0]* or STERM* termination signals.

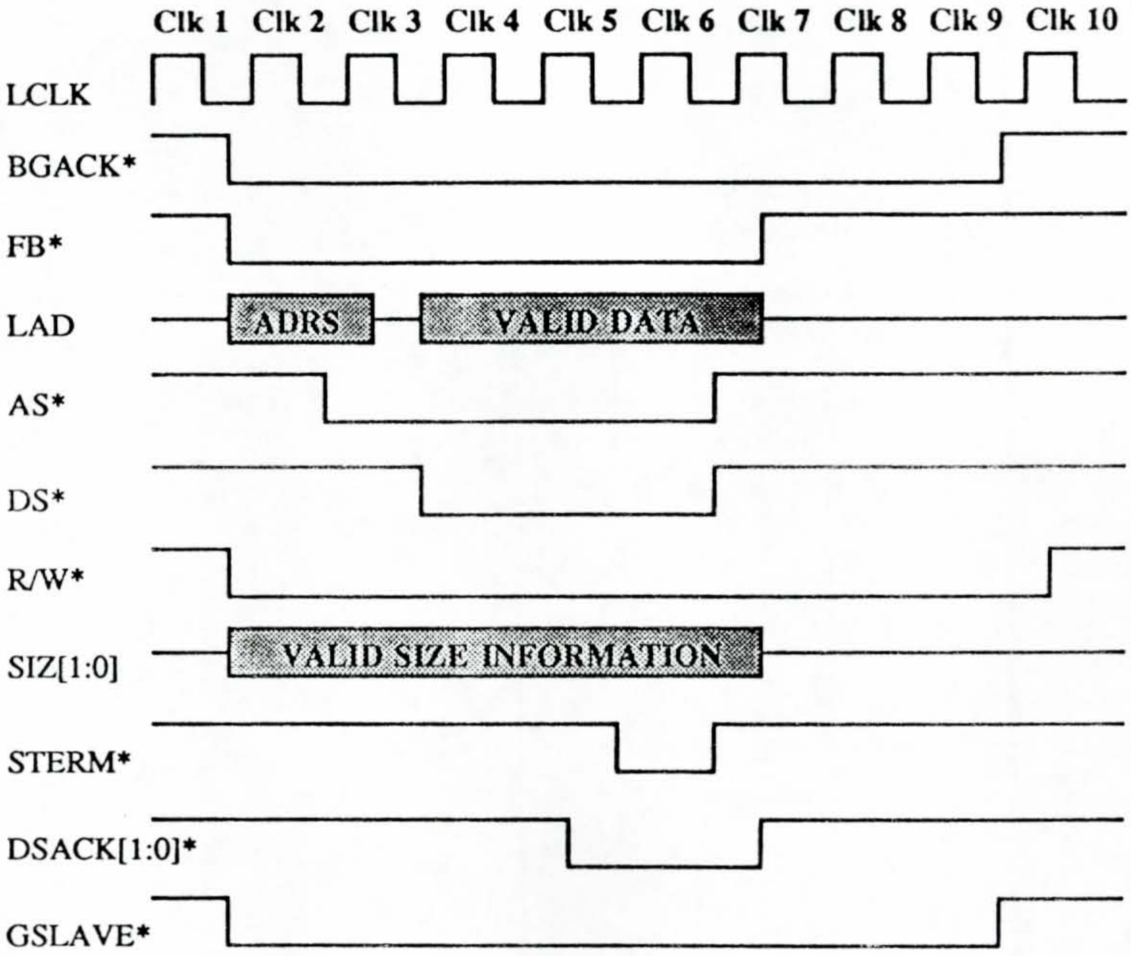
Clock 5: If the DSACK[1:0]* signals are used, then they are asserted during the first phase of the clock. If the signal STERM* is used, then it is asserted during the second phase of the clock. The DSACK[1:0]* signals indicate that the data has been latched while the signal STERM* indicates that the data shall be latched on the next falling edge of the clock.

Clock 6: The local slave latches the data on the falling edge of the clock and deasserts the signal STERM* (if used). The NBIC deasserts the address (AS*) and data (DS*) strobe signals.

Clock 7: Data on the bus becomes invalid and the NBIC deasserts the signals FB*, R/W* and the two size bits (SIZ[1:0]). The local bus slave deasserts the two DSACK* signals during the first phase of the clock (if used).

Clock 9: The NBIC deasserts the signals GSLAVE* and BGACK*. Also, the NBIC stops driving the FB*, R/W* and SIZ[1:0] signals.

Figure 7.3. Single-Word Write to Local Bus Slave



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 6-7, Figure 6-5.

The next transaction to be covered is a single-word read transfer when the NBIC is the local bus master. Figure 7.4 shows the timing diagram for a single-word read transaction with the local bus slave. The sequence of events depicted on the timing diagram are described below:

Clock 1: NBIC is performing bus synchronization during the first phase of the clock. During the second phase of the clock, the NBIC asserts the signals BGACK*, FB*, GSLAVE*, places and address on the bus, drives the signals SIZ[1:0] and R/W* to their appropriate state (drives R/w* high to indicate a read cycle).

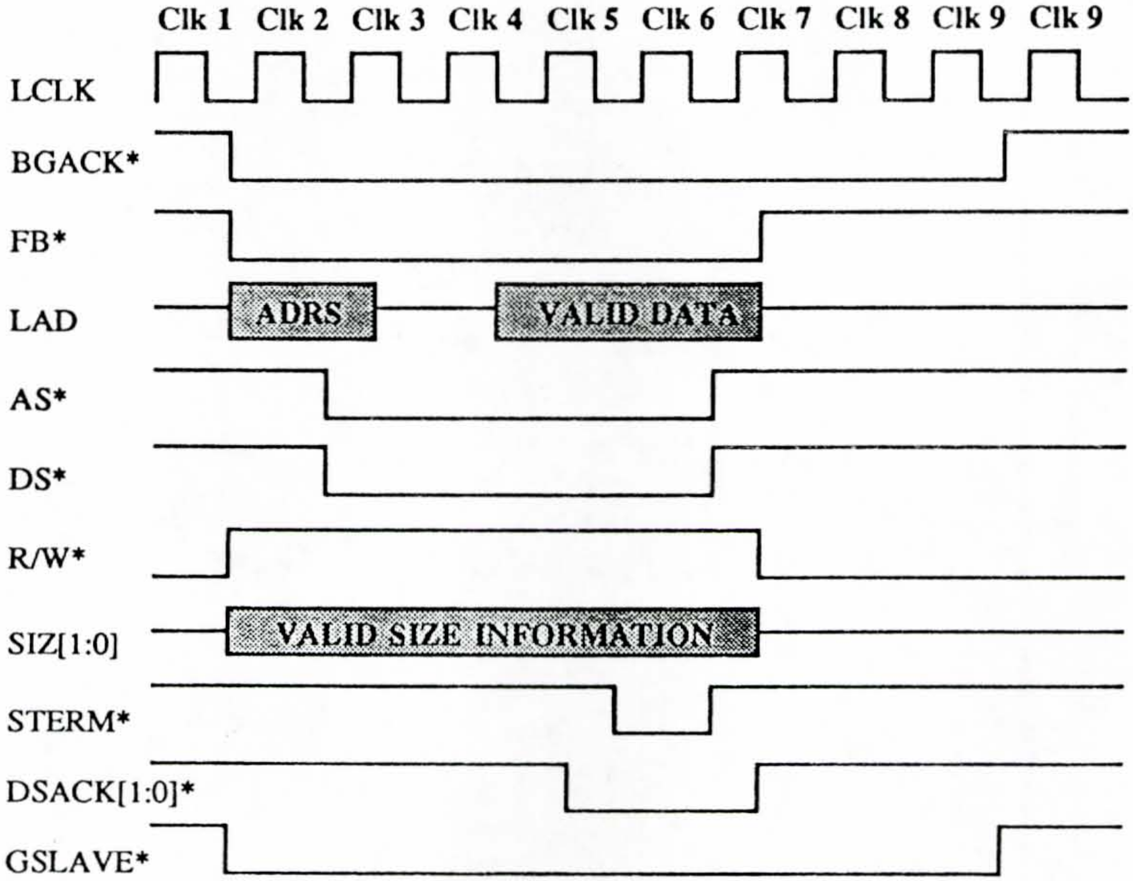
Clock 2: The value on the address/data, R/W* and SIZ[1:0] signal lines are valid during the first phase of the clock. The NBIC asserts the signals AS* and DS*. The local bus slave uses the AS* signal to latch the address, transaction type and size information. The signals AS* and DS* are valid throughout the transaction.

Clock 3: The address becomes invalid during the first phase of the clock.

Clock 4: The NBIC is waiting for the local bus slave to assert one of its termination signals (either DSACK[1:0]* or STERM*).

Clock 5: If the DSACK[1:0]* signals are used, then they are asserted during the first phase of the clock. If the signal STERM* is used, then it is asserted during the second phase of the clock. The signal STERM* indicates that the data shall be valid on the next falling edge of the clock.

Figure 7.4. Single-Word Read from Local Bus Slave



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT, 1989), pg. 6-6, Figure 6-4.

Clock 6: The NBIC latches the data at the trailing edge of the clock and deasserts the signals AS* and DS*. The local bus slave deasserts the signal STERM* (if used).

Clock 7: The NBIC deasserts the signals FB*, R/W* and SIZ[1:0]*. The local bus slave deasserts the signals DSACK[1:0]* (if used) and the data lines during the first phase of the clock.

Clock 9: The NBIC deasserts the signals GSLAVE* and BGACK*. Also, the NBIC stops driving the signals FB*, R/W* and SIZ[1:0].

Burst Transactions on the Local Bus

This section shall cover burst transactions which are initiated by the NBIC when it is a local bus master. Figure 7.5 shows a burst-write timing diagram. The sequence of events depicted in the timing diagram are described below:

Clock 1: During the second phase of the clock, the NBIC asserts the signals BGACK*, FB*, SIZ[1:0] and GSLAVE*. Also, it places an address on the bus and drives the R/W* signal low.

Clock 2: During the first phase of the clock, the values on the data lines, R/W* and SIZ[1:0] signals are valid. Also, the NBIC asserts the burst request (BREQ*) and the address strobe (AS*) signals. The AS* signal remains asserted throughout the transaction.

Clock 3: During the first phase of the clock, the address value is no longer valid. During the second phase of the clock, the NBIC places valid data on the bus and asserts the data strobe signal (DS*) which remains asserted throughout the transaction.

Clock 4: The NBIC is waiting for an acknowledge from the local slave at this time.

Clock 5: During the second phase of the clock, the local bus slave asserts the burst acknowledge (BACK*) signal. Also, the local bus slave asserts the STERM* termination signal which indicates that it shall latch data on the next trailing edge. The signal STERM* remains asserted until all four words have been latched. Recall, that burst transfers consists of four words when using the NBIC. The termination signals DSACK[1:0]* can not be used during burst transactions.

Clock 6: The local bus slave latches data (D1) on the trailing edge of the clock.

Clock 7: The local bus slave latches data (D2) on the trailing edge of the clock.

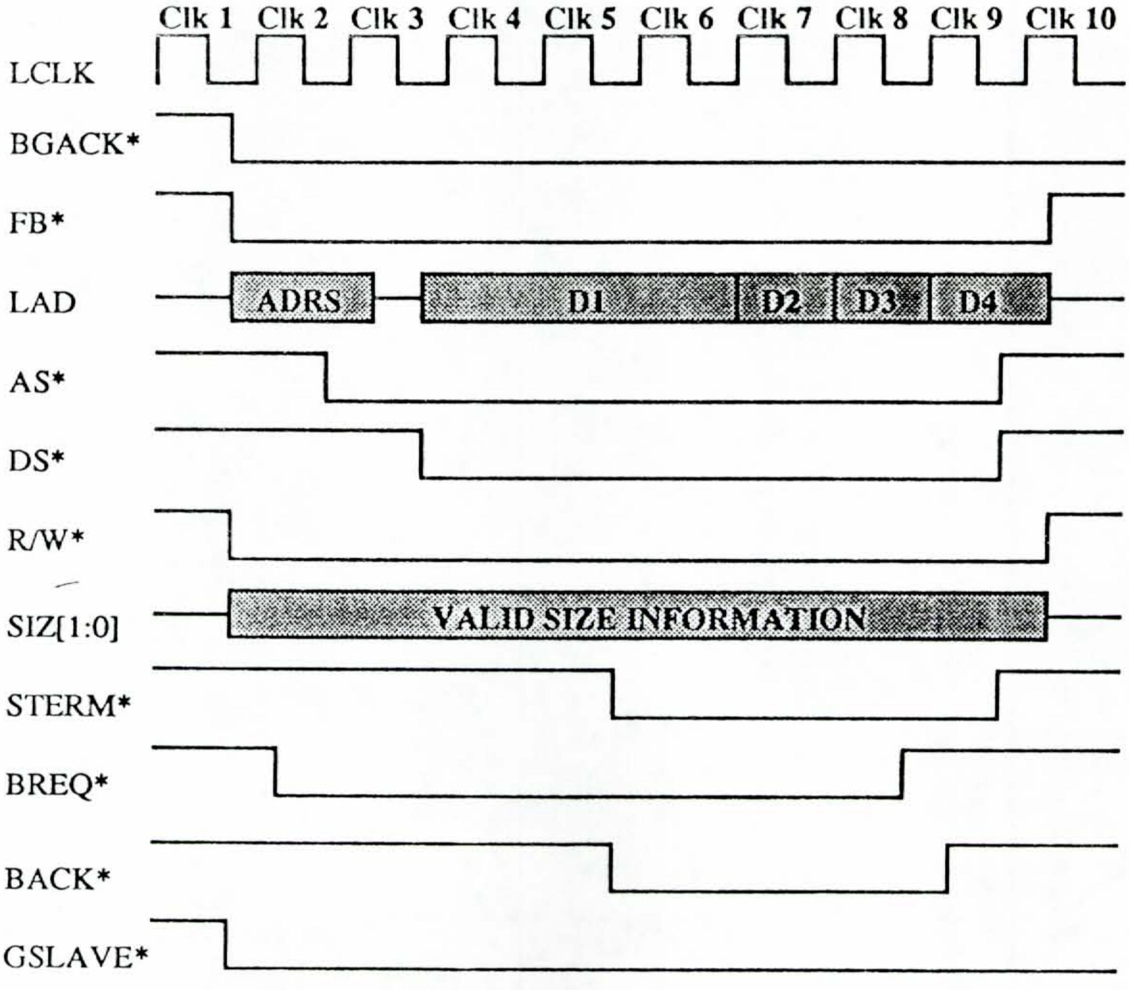
Clock 8: The local bus slave latches data (D3) on the trailing edge of the clock. The

NBIC deasserts the signal BREQ* indicating to the local bus slave that only one word remains to be acknowledged.

Clock 9: During the first phase of the clock, the local bus slave deasserts the signal BACK* indicating that it needs only one more word to complete the transaction. The local bus slave latches the data (D4) on the trailing edge of the clock and deasserts the signal STERM*. Also, the NBIC deasserts both of its strobe signals (AS* and DS*).

Clock 10: During the first phase of the clock, the data becomes invalid and the NBIC deasserts the signals FB*, R/W* and SIZ[1:0].

Figure 7.5. Burst Write to Local Bus Slave.



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", (NeXT 1989), pg. 6-10, Figure 6-7.

The last local bus transaction to be covered in this chapter is a burst read by a local master NBIC. Figure 7.6 shows a timing diagram for a burst read transaction. The sequence of events depicted on the timing diagram are described below:

Clock 1: During the second phase of the clock, the NBIC asserts the signals BGACK*, FB*, SIZ[1:0] and GSLAVE*. Also, it places an address on the bus and drives the signal R/W* high to indicate a read transaction.

Clock 2: During the first phase of the clock, the values on the data lines, R/W* and SIZ[1:0] signals are valid to be sampled by the local bus slave. Also, the NBIC asserts both of its strobe signals (AS* and DS*). The AS* signal is used by the local bus slave to latch the address. The DS* signal informs the local bus slave that it can place data on the bus. Both strobe signals remain asserted throughout the transaction.

Clock 3: During the first phase of the clock the address lines become invalid.

Clock 4: The NBIC is waiting for the local bus slave to assert its termination signal STERM* to indicate that data shall be valid on the next trailing edge.

Clock 5: During the second phase of the clock, the local bus slave asserts the signals BACK* and STERM*.

Clock 6: The NBIC latches data (D1) on the trailing edge of the clock.

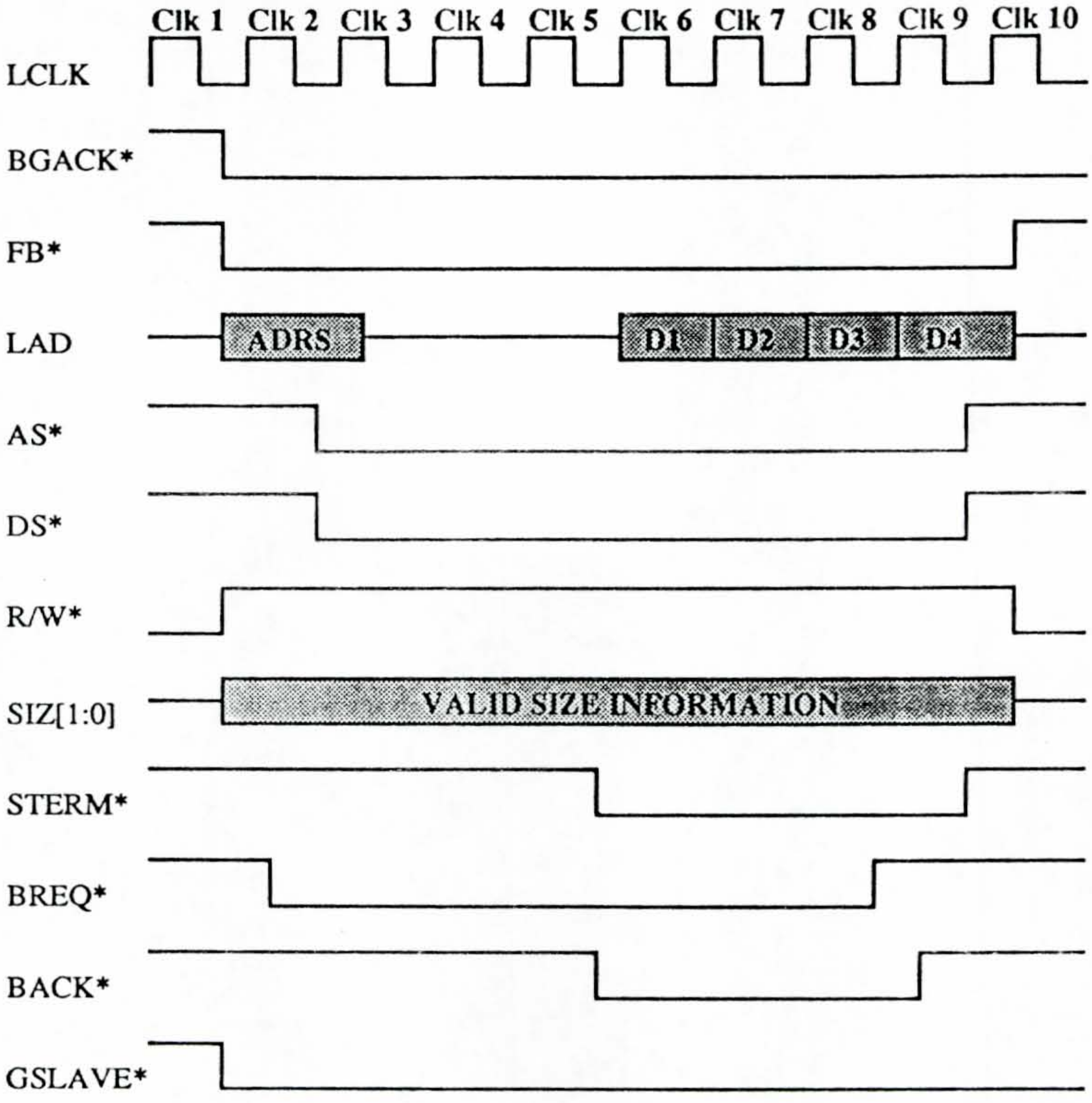
Clock 7: The NBIC latches data (D2) on the trailing edge of the clock.

Clock 8: The NBIC latches data (D3) on the trailing edge of the clock. Also, the NBIC deasserts the signal BREQ* indicating to the slave that it does not want to receive another burst of data.

Clock 9: During the first phase of the clock, the local bus slave deasserts the signal BACK* indicating that the last word is to be transferred. The NBIC latches data (D4) on the trailing edge of the clock. During the second phase of the clock, the NBIC deasserts both of its strobe signals and the local bus slave deasserts the signal STERM*.

Clock 10: During the first phase of the clock, the NBIC deasserts the FB*, R/W* and the SIZ[1:0] signals. Also, the data lines on the bus become invalid during this clock cycle.

Figure 7.6. Burst Read from Local Bus Slave.



SOURCE: Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specifications", (NeXT, 1989), pg. 6-9, Figure 6-6.

CHAPTER VIII

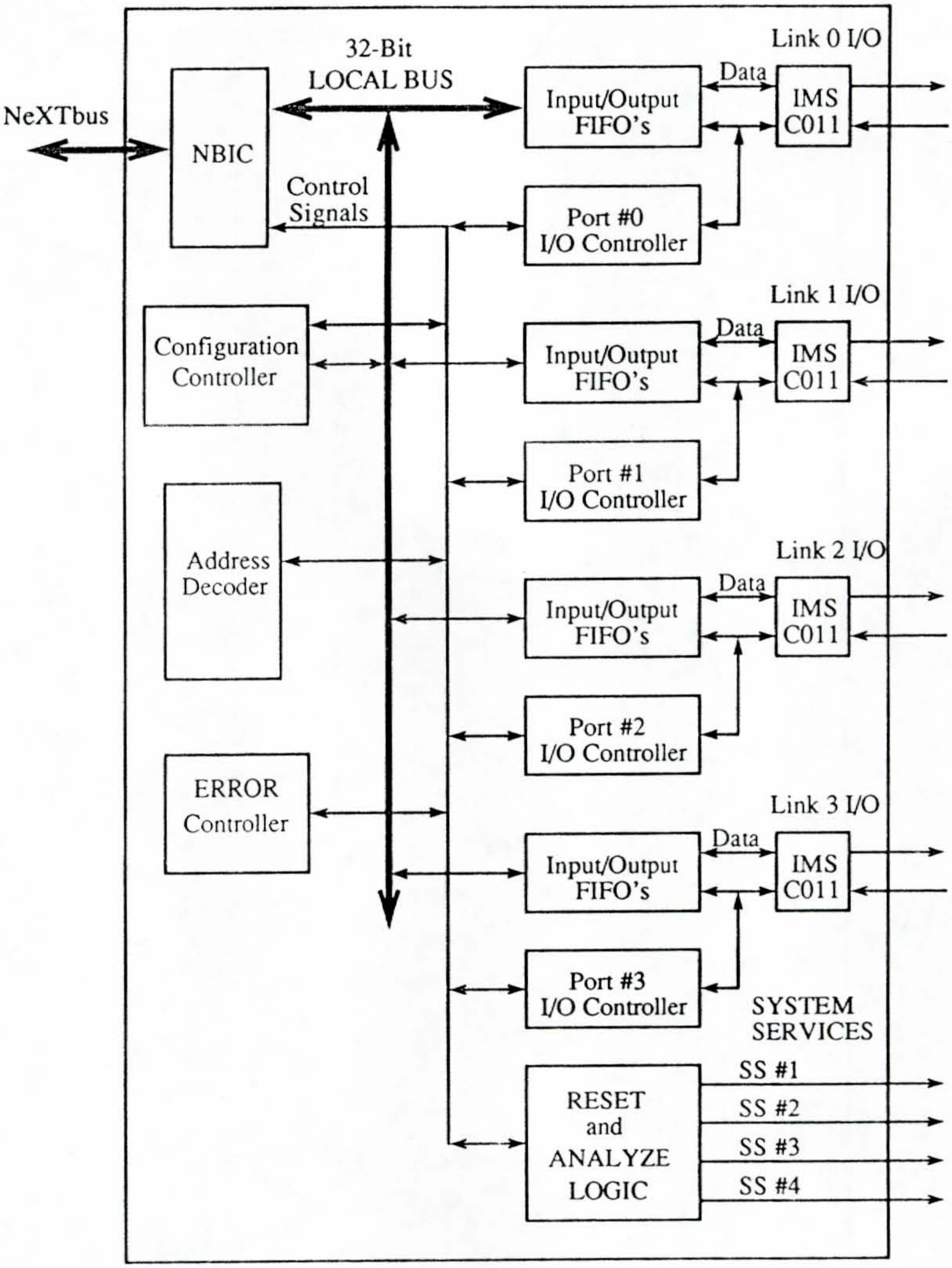
OVERVIEW OF THE INTERFACE BOARD

Introduction to the Board and its Features

The information presented up to this point has introduced the coprocessor architecture, the NeXTbus communication protocol and the NeXTbus Interface Chip. The purpose of this chapter is to introduce the functional blocks of the interface board. The following chapters shall present a top-level design for the functional blocks introduced in this chapter. The functional blocks (modules) introduced in this chapter shall be the basis for the detail design of the interface board.

The NeXT-to-Motherboard interface board shall interface a NeXT computer to an array of coprocessors. This board is a parallel-to-serial/serial-to-parallel quad transputer link interface on a standard NeXT bus card. A top-level block diagram of the board is shown in Figure 8.1. Each port contains bidirectional transputer links with an accompanying set of system services (Reset, Analyze and Error). The ports communicate via the NBIC with the NeXT host processor. Each link port appears to the NeXT host processor as two separate peripherals with the same address space. One of the peripherals processes outgoing data (NeXT-to-Link) to the array of coprocessors and the other peripheral processes incoming data (Link-to-NeXT). The NeXT can transfer data to the ports at a much faster rate that the ports can serially transfer data to the array of coprocessors. However, the NeXT can access only one port at a time while all four ports can be actively transferring data asynchronously with the array of coprocessors. The ports continue to transfer data with the array of coprocessors as long as there is data in the Input FIFO (NeXT-to-Link). To make up for the difference in the two transfer rates, the host processor can cycle from FIFO to FIFO

Figure 8.1. Top-level Block Diagram of Interface Board.

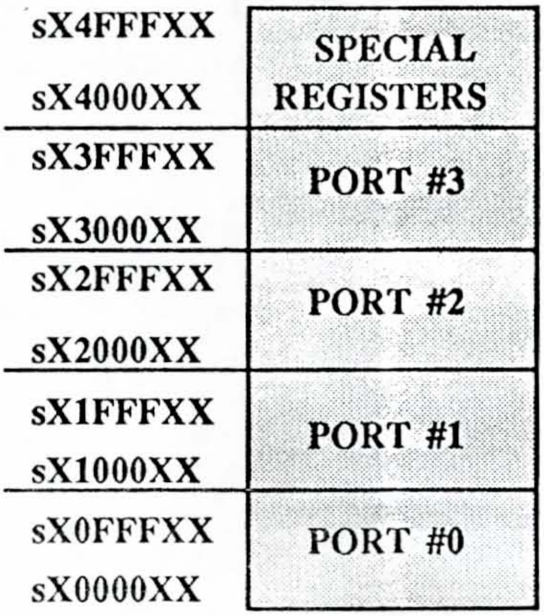


loading or unloading entire blocks of data one at a time. By doing this, the four ports shall be active transmitting the data to the appropriate root transputer in the array of coprocessors. The root transputer is the transputer connected to the host via a link adaptor. All other transputers in the network are connected together via serial links to the root transputer.

Memory Map of Board

The memory map of the board shall define all valid address locations decoded by the board. The memory map shall indicate which port is to process data or if any system services such as a link reset is to be issued. Recall that the NeXTbus addressing range is from s0000000 hex to sFFFFFF hex where s is the slot identification number. Also, the address bits 2E00 through 2E06 have bee predefined by NeXT. The two least significant bits (lsb's) (2E00 - 2E01) are used to encode the type of transaction and the other four bits (2E02 - 2E06) are used to encode the burst size during burst transactions. The board shall use address bits 2E08 - 2E23 to determine which port shall process the data or which special registers to read or write. Figure 8.2 shows the memory map of the board. If the address for a transaction is between sX0000XX through sX0FFFXX, then the data shall be processed by Port #0. The same is true for the remaining addressing ranges shown in Figure 8.2. Each Port address space consists of 4096 (4k) words. The port address space is defined from sXP000XX - sXPFFFXX, where P is the Port number (P = 0 - 3). The upper 256 words (PF00 - PFFF) of each Port are used to enable Burst Byte write transactions. During this type of transaction, only byte 0 (bits[31:24]) shall be written into the Input FIFO. All other burst transactions shall write all four bytes of the 32-bit word. It is important to mention that all Burst read transactions within the appropriate Port address space are 32-bit reads. Table 8.1 shows the special registers which can be access by the user. The special registers provide FIFO status per link and

Figue 8.2. Memory Map of Board



These Address locations
are only valid for Burst Writes.

Burst Byte 3F00 - 3FFF

Burst Byte 2F00 - 2FFF

Burst Byte 1F00 - 1FFF

Burst Byte 0F00 - 0FFF

Table 8.1. Mapping of Special Registers.

PORT #0	PORT #1	PORT #2	PORT #3	DESCRIPTION	Type of Register
				SUBSYTEM SERVICES	
401D	405D	409D	40DD	Subsystem Analyze	Write Only
4019	4059	4099	40D9	Subsystem Reset/Err	Write/Read
				INPUT FIFO (NeXT - to - LINK)	
4015	4055	4095	40D5	FIFO FULL	Read Only
4011	4051	4091	40D1	FIFO EMPTY	Read Only
400D	404D	408D	40CD	FIFO HALF FULL	Read Only
				OUTPUT FIFO (LINK - to - NeXT)	
4009	4049	4089	40C9	FIFO FULL	Read Only
4005	4045	4085	40C5	FIFO EMPTY	Read Only
4001	4041	4081	40C1	FIFO HALF FULL	Read Only

provide the user the capability to reset or analyze a node of the coprocessor. A write to the subsystem reset/error register shall reset the root transputer connected to that particular Port. A read to the subsystem reset/error register shall provide the error flag of the root transputer subsystem. All of the special registers are one bit in size (bit 0). Each pair of links can be used to configure or analyze different nodes in the array of coprocessors.

Different Functional Modules on the Board

The board is composed of five functional modules. Each module shall be briefly described in this section. The following chapters shall provide a top-level functional design of each of the modules. The modules which provide the basis for the detail design of the interface board are the configuration controller, address decoder, port controller(s), error controller and reset logic.

The configuration controller is only executed during the power on sequence or when the reset signal (LRESET*) is asserted by the NBIC. It shall initialize all the NBIC internal registers such as the Control and NBIC Identification registers. Two different functional designs shall be presented in Chapter IX. The board's designer shall have the flexibility of selecting one of the designs presented or derive a new design based on the ideas presented. Also, this controller shall contain the external arbiter of the board. The arbiter shall grant the local bus to the NBIC when requested by asserting the bus grant signal (BG*). The board's logic shall only have ownership of the local bus while configuring the NBIC internal registers. All other times, the NBIC shall receive a bus grant (BG*) signal from the controller when it asserts its bus request (BR*) signal.

The address decoder module shall latch the address of a transaction and set the appropriate Port control signals. The signals asserted by the address decoder are used by the Port controller and error controller to determine which sequence of states to execute. This module shall assure that only one Port Controller is involved in a transaction with the host processor at any given time.

The Port controller shall provide all the control signals to communicate with the NBIC and with the Inmos IMS C011 link adaptor. This controller actually consists of two independent port controllers (Input and Output). The Input port controller interfaces with the NBIC and writes data into the Input FIFO (NeXT-to-Link) and reads data from the Output FIFO (Link-to-NeXT). Also, this controller shall provide the control signals required to read the status flags from the appropriate port FIFO or the error flag from the subsystem. The output port controller interfaces with the IMS C011 link adaptor. It shall read data from the Input FIFO and write data into the Output FIFO.

The error controller shall be responsible for generating a bus error (BERR*) transaction termination when the user attempts to access an invalid memory address of the interface board or access a valid memory location incorrectly. This controller terminates an invalid transaction so that the NeXT bus master does not have to wait until clock cycle 256 for a bus timeout error to occur. A bus timeout error shall be issued by the NeXT CPU board in slot 0 if no termination is generated by cycle 256.

The reset module shall provide all the reset and analyze signals to the appropriate FIFO's, link adaptor and root transputers. This module shall be defined in terms of a timing diagram. The user shall have the freedom of implementing this module any way desired. There are timing restrictions for resetting the root transputers and link adaptors which must be met. Also, this logic shall provide the transputer analyze timing sequence on a port basis when requested by the user. The analyze sequence is used to determine the state of a transputer based network. Finally, this module shall also be responsible for generating the local interrupt signal (SINT*). When the signal SINT* is asserted by this module and the mask interrupt register is enabled, then an interrupt signal shall be generated onto the NeXTbus by the NBIC.

CHAPTER IX

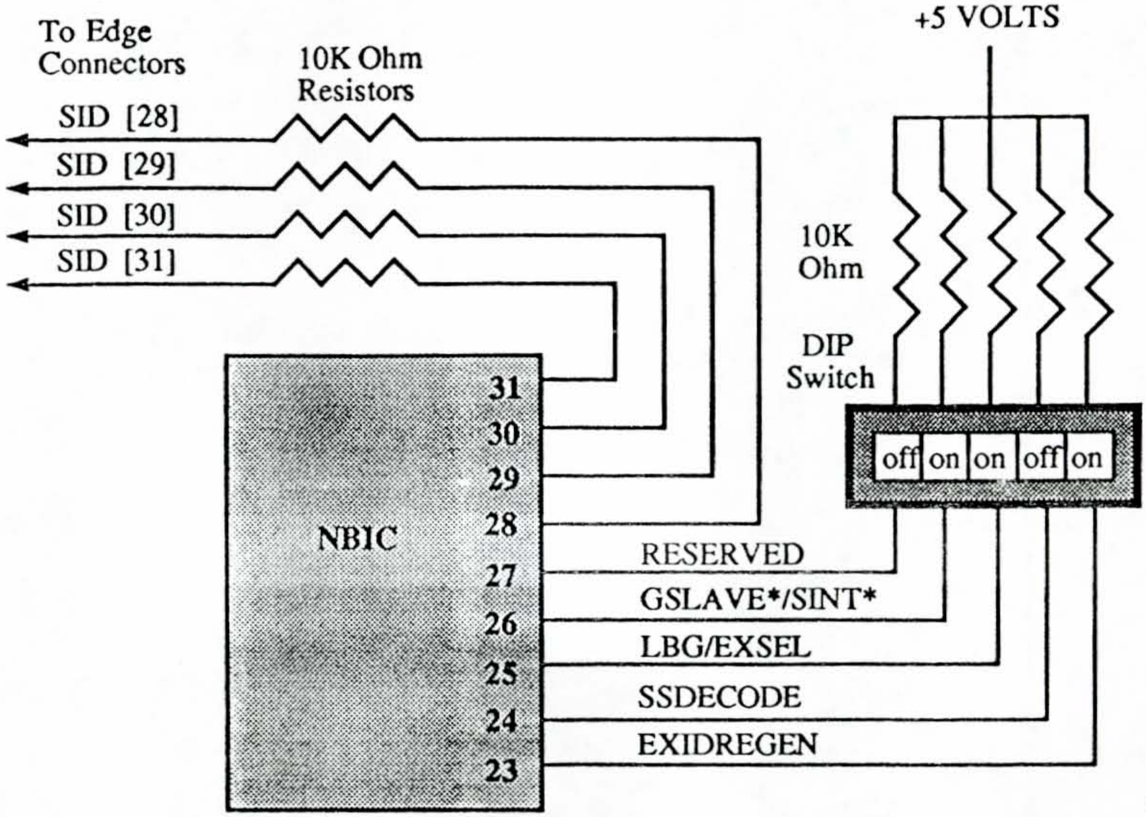
TOP-LEVEL DESIGN OF CONFIGURATION CONTROLLER

The purpose of this chapter is to provide a functional top-level design for the configuration controller. This controller shall initialize the NBIC identification register (ID) and the control register. Recall, that the configuration register is configured by board resistors. The configuration of this register shall be defined before the design of the configuration controller is presented.

The configuration register shall be setup to select the signals SINT*, EXSEL and EXIDREGEN bits 26, 25 and 23, respectively. These bits are selected by placing 10,000 Ohm resistors from the appropriate bits to +5 volts. The signal SINT* shall be asserted by the board's logic to generate a NeXTbus global interrupt signal (GINT*). The EXSEL signal shall be asserted by the NBIC when a NeXTbus master references the slot address space or ID register. The EXIDREGEN bit is enabled since the NBIC ID register shall be contained inside the NBIC instead of the board's memory space. The SSDECODE bit shall not be selected which cause a bus timeout error for any references to the board's slot address space (does not include ID or Interrupt registers). A dip switch shall be used to select the different bits of the configuration registers. A block diagram of the NBIC with the configuration resistors is shown in Figure 9.1.

The configuration controller shall interface with the NBIC, a byte size PROM, four registers with output and clock enables and one quad register. The PROM shall be used to store the address and data for the NBIC ID and control registers. The registers with clock enables shall be used to store the appropriate bytes of the 32-bit ad-

Figure 9.1. NBIC Configuration Register Selection.



dress or data. The quad register shall be used to sample the negative edge signals. The controller shall be clocked by the positive edge of LCLK while the quad D flip-flop shall be clocked by an inverse LCLK (or trailing edge of LCLK). A top-level block diagram of the hardware required for the configuration sequence is shown in Figure 9.2. The contents of the PROM are shown in Figure 9.3. The state-diagram of the controller is shown in Figure 9.4. The AHPL description of the controller is shown in Figure 9.5. Finally, a timing diagram for the configuration sequence is shown in Figure 9.6. The configuration controller begins to execute after an LRESET* pulse (approx. 1.28 us) is received from the NBIC. The sequence of events depicted in the timing diagram are described below:

LCLK #1 : The signal LRESET* is asserted by the NBIC.

Figure 9.2. Top-level Diagram of the Configuration Controller Hardware Architecture.

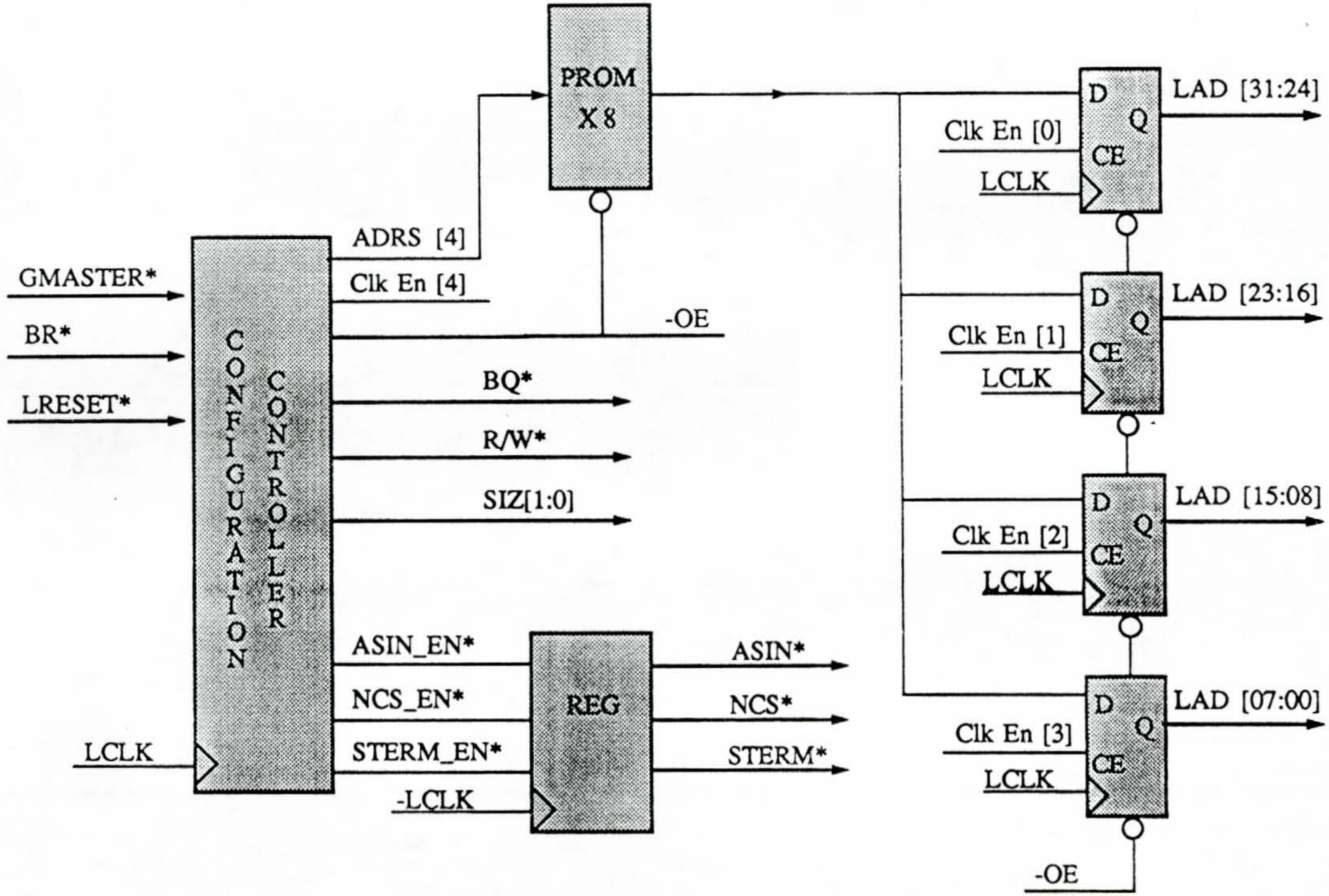


Figure 9.3. Contents of PROM in Configuration Architecture.

ADRS	
00	'04' hex LSB Adrs of NBIC ID Reg
01	'00' hex Remaining Adrs for NBIC ID
02	NBIC ID LSB Byte #0
03	NBIC ID Byte #1
04	NBIC ID Byte #2
05	NBIC ID MSB Byte #3
06	'00' hex LSB Adrs of Control Reg
07	'00' hex Remaining Adrs for Control Reg
08	Control Register Byte #0
09	Control Register Byte #1
0A	Control Register Byte #2
0B	Control Register Byte #3

LCLK #2 : The configuration controller is initialized to State Number #01.

LCLK #3 : The controller goes to State Number #02. While in this state, the PROM address is zero and the clock enable lsb is set (CNT_EN[3] = 1). AHPL notation uses a Big-Endian byte ordering scheme. (CNT_EN <-- 4 T 1 " SET CNT_EN[3] = 1).

LCLK #4 : During the rising edge of this clock, the lsb byte of the NBIC ID register address is latched. The PROM address equals one and the three msb's of the clock enables are set.

LCLK #5 : During the rising edge of the clock, the remaining bytes of the NBIC ID register address are latched. Also, the PROM address equals two and the lsb clock enable is set. During the trailing edge of the clock, the signal ASIN* is asserted.

LCLK #6 : The configuration controller sequences to State Number #05 where the PROM address equals three and the next byte of the clock enable is set. During the rising edge, the lsb byte of the NBIC ID register data is latched. During the trailing edge of the clock, the signal NCS* is asserted to indicate an internal register transaction.

LCLK #7 : The configuration controller sequences to State Number #06. During the rising edge of the clock, the next byte of the NBIC ID register is latched.

LCLK #8 : The configuration controller sequences to State Number #07. During the rising edge of the clock, the next byte of the NBIC ID register is latched.

Figure 9.4. State-Diagram for Configuration Controller.

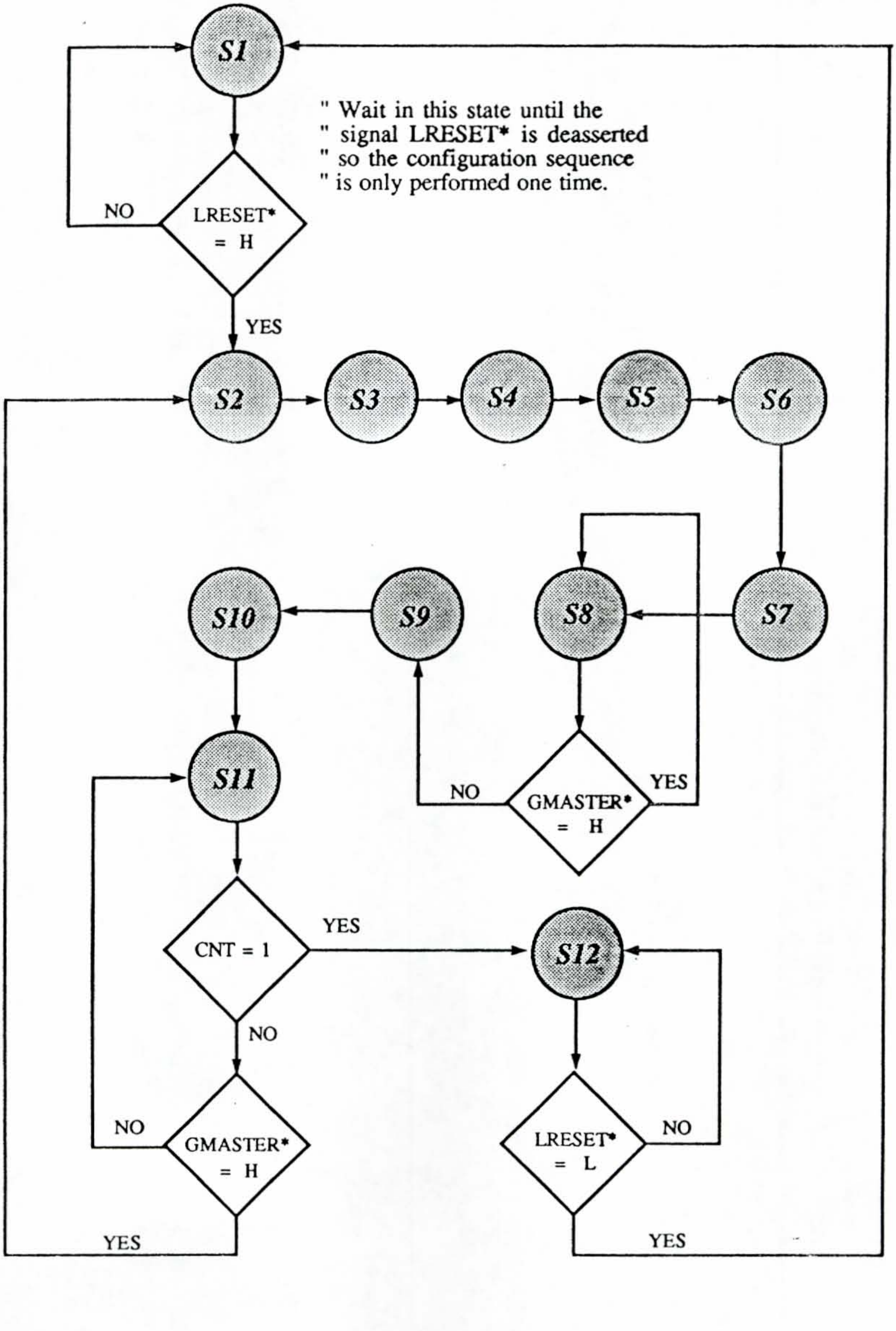

S1
" Wait in this state until the
" signal LRESET* is deasserted
" so the configuration sequence
" is only performed one time.
NO
LRESET* = H
YES
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
NO
GMASTER* = H
YES
CNT = 1
YES
NO
S12
NO
GMASTER* = H
YES
LRESET* = L
NO
YES

Figure 9.5. AHPL Description for Configuration Controller.

```
MODULE:  CONFIGURATION_CONTROLLER
MEMORY:  CNT[1].
INPUTS:   GMASTER*; LRESET*; BR*.
OUTPUTS: R/W*; SIZ [2]; ADRS [4]; CLK_EN [4]; OE*;
          ASIN_EN*; NCS_EN*; STERM_EN*; BG*.

" Description of Inputs
"
" GMASTER*       = NBIC asserts this signal to inform the configuration
"                   controller that it is performing as a local bus slave.
" LRESET*        = Local reset signal from NBIC (approx. 1.28 us).
" BR*            = This is an open-drain wired "OR" signal which is
"                   asserted by the NBIC when it wants ownership
"                   of the board's local bus.
"
" Description of Outputs
"
" R/W*           = Asserted low by configuration controller when in the
"                   process of writing to NBIC otherwise tri-stated.
" SIZ[1:0]       = Size transfer bits which indicate number of valid bytes.
" ADRS[4]        = Address to the PROM external to configuration controller.
" CLK_EN[4]      = Chip (byte) enables to the 32-bit byte loadable register.
" OE*            = Output enable for the 32-bit byte loadable register.
" ASIN_EN*       = Enable signal for the Address Strobe signal to indicate
"                   to the NBIC that a valid address is on the bus.  The
"                   actual signal (ASIN*) is clocked external on a -LCLK.
" NCS_EN*        = Enable signal for the NBIC chip select signal (NCS*)
"                   which is asserted by controller to indicate to the NBIC
"                   that an internal register transaction is to take place.
" STERM_EN*      = Enable signal for the synchronous termination signal.
" BG*            = Bus Grant from the controller when configuration is complete.


BODY

1. ---> ( LRESET*)/ ( 1 );  ADRS <--- 4 T 0;          " Stay in this state till
    CNT <--- 0; CLK_EN <--- 4 T 1;  R/W* = Z;         " the signal LRESET*
    OE* = 1;  BG* = 1;  SIZ <--- 2 T Z.               " goes High. ( Z = tri-state)


2. ADRS <--- INC(ADRS); OE* = 0;                      " Increment Address to
   CLK_EN <--- 4 T 14;  R/W* = 0 ;                    " PROM.  Enable output
   SIZ[2] = 2 T 0;  BG* = 1.                          " enable & set OE* low.


3. ADRS <--- INC(ADRS); OE* = 0;                      " Increment PROM address
   CLK_EN <--- 4 T 1; R/W* = 0;                       " Deassert all the trailing
   SIZ[2] = 2 T 0; BG* = 1;                           " Edge signals.
   ASIN_EN* = 1; NCS_EN* = 1; STERM_EN* = 1.
```

Figure 9.5. Con't.

```
 4.  ADRS <--- INC(ADRS); OE* = 0;                      " Assert ASIN_EN* to
     CLK_EN <--- 4 T 2; R/W* = 0;                       " indicate to NBIC that the
     SIZ[2] = 2 T 0; BG* = 1;                           " Address is valid.
    ASIN_EN* = 0; NCS_EN* = 1; STERM_EN* = 1.

 5.  ADRS <--- INC(ADRS); OE* = 0;                      " Assert NCS_EN* to
     CLK_EN <--- 4 T 4; R/W* = 0;                       " indicate to NBIC that the
     SIZ[2] = 2 T 0; BG* = 1;                           " transaction is to an internal
    ASIN_EN* = 0; NCS_EN* = 0; STERM_EN* = 1.           " register.

 6.  ADRS <--- INC(ADRS); OE* = 0;                      " ASIN_EN* and NCS_EN*
     CLK_EN <--- 4 T 4; R/W* = 0;                       " remain asserted throughout
     SIZ[2] = 2 T 0; BG* = 1;                           " the transaction.
    ASIN_EN* = 0; NCS_EN* = 0; STERM_EN* = 1.

 7.  ADRS <--- INC(ADRS); OE* = 0;                      " Next clock kill all clock
     CLK_EN <--- 4 T 0; R/W* = 0;                       " Enables since all the NBIC
     SIZ[2] = 2 T 0; BG* = 1;                           " ID registers byte have been
    ASIN_EN* = 0; NCS_EN* = 1; STERM_EN* = 1.           " accessed from the PROM.

 8.  ---> ( GMASTER*)/ ( 8 );  OE* = 0;                 " This state might not be required.
      R/W* = 0; BG* = 1; SIZ <--- 2 T 0;                " It is included in the design
      ASIN_EN* = 0; NCS_EN* = 0; STERM_EN* = 1;         " as a safety measure to verify
                                                        " that NBIC has responded to
                                                        " the NCS* signal.

 9.  OE* = 0; R/W* = 0; BG* = 1; SIZ <--- 2 T 0;        " Assert  STERM_EN*
     ASIN_EN* = 0; NCS_EN* = 0; STERM_EN* = 0.          " to indicate to NBIC to latch
                                                        " data on next trailing edge.

10.  OE* = 0; R/W* = 0; BG* = 1; SIZ <--- 2 T 0;        " Deassert STERM_EN*
     ASIN_EN* = 1; NCS_EN* = 1; STERM_EN* = 1.          " to indicate to NBIC that
                                                        " data is no longer valid.

11.  OE* = 1; R/W* = Z; BG* = 1; SIZ <--- 2 T Z;        " Tri-State R/W*, and the two
      CNT * (GMASTER) <--- 1 ;                          " Size transfer bits.
      $---> (CNT, GMASTER \wedge \overline{CNT}, \overline{GMASTER* \wedge CNT})/( 12, 1, 11 ).$

12.  OE* = 1;  R/W* = Z; SIZ <--- 2 T Z;                " This state acts as an external
     BG* = BR*;                                         " arbiter.  Signal bus grant is
     --> ( LRESET*)/(12);                               " asserted when a bus request
     ASIN_EN* = 1; NCS_EN* = 1.                         " is received  from the NBIC.

END SEQUENCE

     $---> (\overline{LRESET*})/( 1 ).$                  " Go to State 1 if there is a reset
                                                        " pulse from the NBIC.

END.
```

LCLK #9 : The configuration controller sequences to State Number 08. During the rising edge of the clock, the msb byte of the NBIC ID register is latched. The clock enables are all zero. The controller stays in this state until the signal GMASTER* is low. If GMASTER * is low, then the controller shall sequence to State Number #09.

LCLK #10 : During the trailing edge of the clock, the signal STERM* is asserted to indicate to the NBIC to latch data on the next trailing edge of the clock.

LCLK #11 : During the trailing edge of the clock, the signals STERM*, ASIN* and NCS* are deasserted.

LCLK #12 : The controller sequences to State Number #11 where it waits for the signal GMASTER* to be deasserted by the NBIC.

LCLK #13 : The signal GMASTER* is not deasserted so the controller stays in State Number 11.

LCLK #14 : The signal GMASTER* is not deasserted so the controller stays in State Number 11.

LCLK #15 : The signal GMASTER* is deasserted and the controller sequences to State Number 02. The address in this state is six which is the lsb byte of the Control register address in the PROM. The signal CNT is set during the transition.

LCLK #16 : During the rising edge of the clock, the lsb byte of the Control register address is latched.

LCLK #17 : During the rising edge of the clock, the remaining three msb bytes of the Control register address is latched. During the trailing edge of the clock, the signal ASIN* is asserted.

LCLK #18 : During the rising edge of the clock, the NBIC latches the address and the lsb byte of the Control register data is latched. During the trailing edge of the clock, the signal NCS* is asserted.

LCLK #19 : During the rising edge of the clock, the next byte of the Control register is latched.

LCLK #20 : During the rising edge of the clock, the next byte of the Control register is latched.

LCLK #21 : During the rising edge of the clock, the msb byte of the Control register is latched. The controller sequences to State Number #08 during this edge.

LCLK #22 : The controller sequences to State Number #09 since the signal GMASTER* is asserted by the NBIC. During the trailing edge of this clock, the signal

STERM* is asserted.

LCLK #23 : The Control register data is latched during the trailing edge of the clock and the signals STERM*, ASIN* and NCS* are deasserted.

LCLK #24 : The controller sequences to State Number #11.

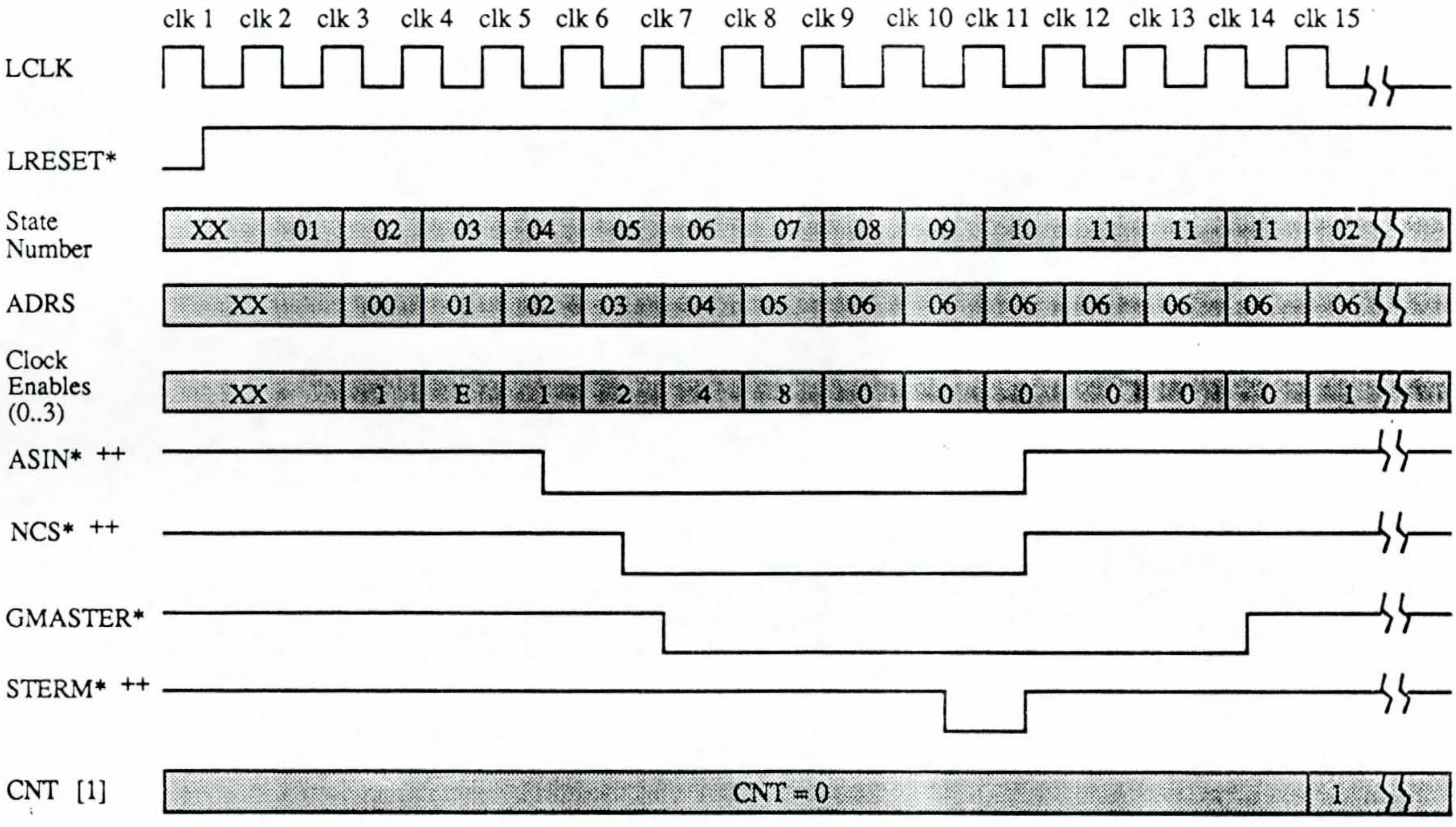
LCLK #25 : The controller sequences to State Number #12 since the signal CNT is set. This indicates that the configuration sequence is complete. While in State Number 12, the controller acts as the external arbiter. When the NBIC asserts the bus request (BR*) signal, the controller shall assert the bus grant signal (BQ*). The R/W*, SIZ[1:0], Address/Data lines and STERM* signals shall be tri-stated. The signals ASIN_EN*, CLK_EN[4] and NCS_EN* shall remain deasserted in this state.

LCLK #25 - #29: The controller remains in State Number #12 until a reset signal (LRESET*) is received from the NBIC. At this time, the configuration sequence shall repeat its sequence beginning with State #1. The controller shall remain in State #1 until the reset pulse is deasserted. The reason being that the LRESET* pulse is approximately 1.28 microseconds and the controller can cycle through all its states finishing up in State #12 before the reset pulse goes high. If the controller is in State #12 and the reset pulse is still asserted (LOW), then the configuration cycle will execute again; therefore, the configuration cycle shall begin when the reset pulse (LRESET*) is deasserted.

The signal GMASTER* is used in this controller as an acknowledgment from the NBIC that it is performing as a local bus slave. This is the only time when the NBIC shall be a slave on the local bus. All transactions on the local bus shall be initiated by the NBIC during all other transactions. The configuration controller shall grant the bus to the NBIC after the sequence is complete by asserting the signal BG*.

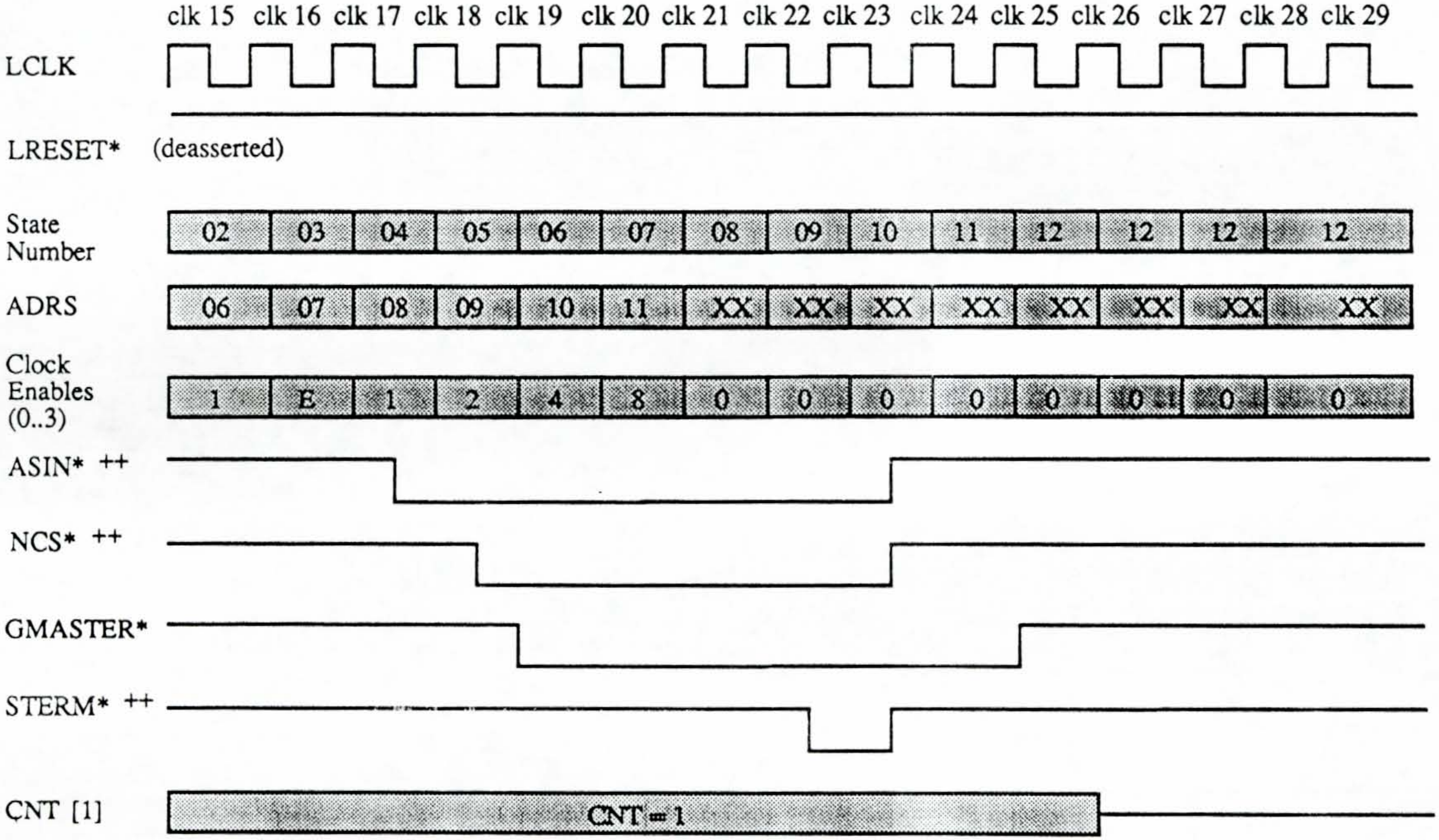
An alternative method is to use four registered PROM with output enables such as the Cypress CY7C225. This PROM is a 512 X 8 registered PROM with output enables. The advantage of this method is that there are less cycles required to build the 32-bit address and data word for the registers. The disadvantage is that there are four programmable parts on the board and only four locations shall be valid in each PROM. The design for this method shall not be presented. The designer can use one of the two methods presented or derive a new implementation for the configuration sequence from the ideas presented in this chapter.

Figure 9.6. Timing Diagram for Configuration Sequence.



NOTES: ++ These signals are sampled at the trailing edge of LCLK. CLOCK ENABLES = E represents CLK_EN[0:2] = 1.

Figure 9.6. Con't.

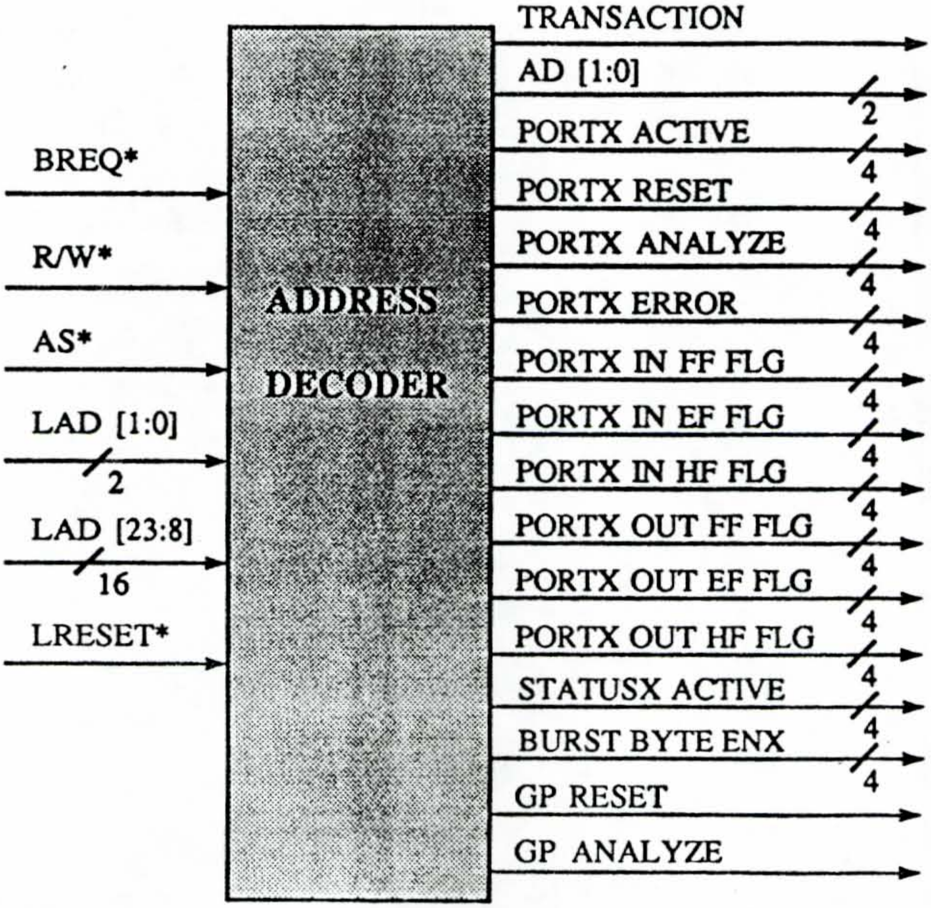

clk 15 clk 16 clk 17 clk 18 clk 19 clk 20 clk 21 clk 22 clk 23 clk 24 clk 25 clk 26 clk 27 clk 28 clk 29
LCLK
LRESET* (deasserted)
State Number
02 03 04 05 06 07 08 09 10 11 12 12 12 12
ADRS
06 07 08 09 10 11 XX XX XX XX XX XX XX XX
Clock Enables (0..3)
1 E 1 2 4 8 0 0 0 0 0 0 0 0
ASIN* ++
NCS* ++
GMASTER*
STERM* ++
CNT [1]
CNT = 1

CHAPTER X

TOP-LEVEL DESIGN OF ADDRESS DECODER

The purpose of this chapter is to provide a functional top-level design for the address decoder module. This module shall latch the address at the beginning of a transaction and set the appropriate PORT control signals. A top-level block diagram of the address decoder is shown in Figure 10.1. A state-diagram representing the sequence of events is shown in Figure 10.2. The AHPL description for the

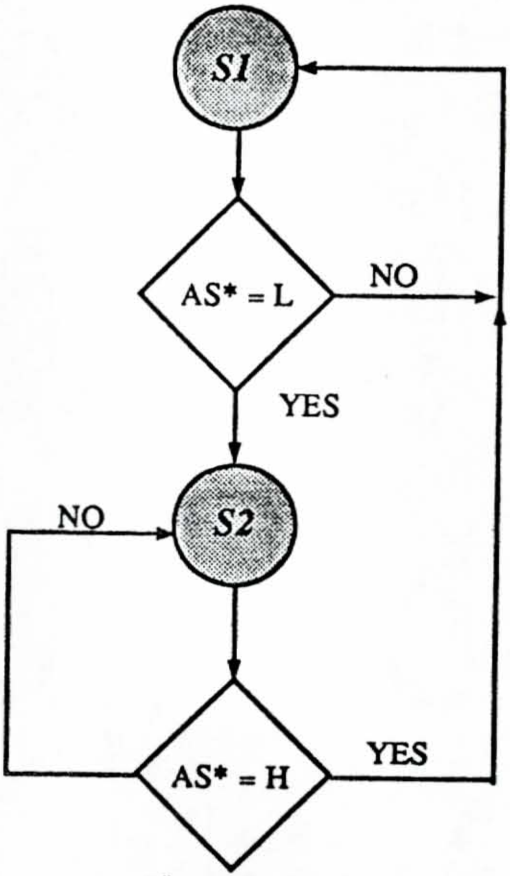
Figure 10.1. Top-level Block Diagram of Address Decoder.



Where X is the PORT NUMBER (0 - 3)

address decoder module is shown in Figure 10.3. The module shall latch the address at the first rising edge of LCLK after the signal AS* is asserted by the NBIC. The

Figure 10.2. State-Diagram for Address Decoder.



PORT signals shall be set according to the memory map of the board shown in Table 8.1 and Figure 8.2. The signal denoted by TRANSACTION shall be set by the address decoder when it is in State 2. This signal is used to alert the Error Controller that a NeXTbus transaction is starting.

The R/W* and BREQ* signals are used by the address decoder to determine which PORT signals to assert. For example, if the address bits 23 through 8 equal '4019' hex, then the R/W* signal is used to determine which of the two possible signals to assert (PORT0_RESET or PORT0_ERROR). If the signal R/W* is high,

Figure 10.3. AHPL Description for Address Decoder.

```
MODULE:  ADDRESS DECODER
MEMORY:  ADRS [16], RESET, ANALYZE.
INPUTS:    LAD[23:8]; LAD[1:0]; AS*; R/W*, BREQ*, LRESET*.
OUTPUTS: PORT0_ACTIVE, PORT1_ACTIVE, PORT2_ACTIVE;
         PORT3_ACTIVE; PORT0_ANALYZE;
         PORT1_ANALYZE; PORT2_ANALYZE;
         PORT3_ANALYZE; PORT0_RESET; PORT1_RESET;
         PORT2_RESET; PORT3_RESET; PORT0_ERROR;
         PORT1_ERROR; PORT2_ERROR; PORT3_ERROR;
         PORT0_IN_FF_FLG;  PORT1_IN_FF_FLG;
         PORT2_IN_FF_FLG;  PORT3_IN_FF_FLG;
         PORT0_IN_EF_FLG;  PORT1_IN_EF_FLG;
         PORT2_IN_EF_FLG;  PORT3_IN_EF_FLG;
         PORT0_IN_HF_FLG;  PORT1_IN_HF_FLG;
         PORT2_IN_HF_FLG;  PORT3_IN_HF_FLG;
         PORT0_OUT_FF_FLG;  PORT1_OUT_FF_FLG;
         PORT2_OUT_FF_FLG;  PORT3_OUT_FF_FLG;
         PORT0_OUT_EF_FLG;  PORT1_OUT_EF_FLG;
         PORT2_OUT_EF_FLG;  PORT3_OUT_EF_FLG;
         PORT0_OUT_HF_FLG;  PORT1_OUT_HF_FLG;
         PORT2_OUT_HF_FLG;  PORT3_OUT_HF_FLG;
         BURST_BYTE_EN0, BURST_BYTE_EN1;
         BURST_BYTE_EN2, BURST_BYTE_EN3;
         STATUS0_ACTIVE, STATUS1_ACTIVE;
         STATUS2_ACTIVE, STATUS3_ACTIVE;
         TRANSACTION; AD[2]; GP_RESET; GP_ANALYZE.

" Description of Inputs

" LAD[1:0]           = Address lines which carry transaction information.
" LAD[23:8]          = Address lines which are used to decode the type
"                      of transaction to be performed.
" AS*                = Address Strobe signal from NBIC.
" R/W*               = Read = 1 / Write = 0 (From NBIC).
" BREQ*              = Burst Request signal from NBIC.
" LRESET*            = Reset Pulse from NBIC.
" GP RESET           = Asserted when any port is to be reset.
" GP ANALYZE         = Asserted when any port is to be analyzed.

" Description of Outputs

" TRANSACTION        = Set High when in State 2 and not performing a
"                      PORT Reset or Analyze sequence.
" AD[1:0]            = The value of the two LSB's of the address lines.
" PORTX ACTIVE       = Set High when the address is within the memory
                       map limits of the appropriate PORT (Read or Write).
" PORTX RESET        = Set High when a Write transaction is performed to
                       the appropriate PORT address to Reset that PORT.
```

Figure 10.3. Con't.

```
" PORTX ANALYZE       = When set used by the Reset Module to
"                        send appropriate analyze signals to network.
" PORTX ERROR         = When set used by Input Port Controller to place
"                        the status of the appropriate PORT Error Flag
"                       on the LSB of the address line.
" PORTX IN FF FLG     = Set High when the user request the status of the
"                       FIFO FULL flag from the Input FIFO PORTX.
" PORTX IN EF FLG     = Set High when the user request the status of the
"                       EMPTY FIFO flag from the Input FIFO PORTX.
" PORTX IN HF FLG     = Set High when the user request the status of the
"                       FIFO HALF FULL flag from the Input FIFO PORTX.
" PORTX OUT FF FLG    = Set High when the user request the status of the
"                       FIFO FULL flag from the Output FIFO PORTX.
" PORTX OUT EF FLG    = Set High when the user request the status of the
"                       EMPTY FIFO flag from the Output FIFO PORTX.
" PORTX OUT HF FLG    = Set High when the user request the status of the
"                       FIFO HALF FULL flag from the Output FIFO PORTX.
" BURST_BYTE_ENX      = Set by Address Decoder when a Burst Byte Transaction
"                       is to take place.  (Only byte 0 of each word is written).
" STATUSX_ACTIVE      = Set by Address Decoder when the user request one
"                       of the possible seven status flags (on a per port basis).
" GP RESET            = Asserted when any port is to be reset.
" GP ANALYZE          = Asserted when any port is to be analyzed.
```

BODY

```
1.  ADRS * ($\overline{AS*}$) <---  LAD [23:8];           " Latch address lines from
    AD[0:1] * ($\overline{AS*}$) <--- LAD[0:1];           " NBIC and go to State 2
    --->   ($\overline{AS*}$)/(2).                         " when AS* goes low.

                                                          " Decode all valid
2.  TRANSACTION      = 1 ;                                " address locations
    PORT0_ACTIVE     = ( ADRS[0:3] = 4 T 0 );             " Port 0 Read/Write
    PORT1_ACTIVE     = ( ADRS[0:3] = 4 T 1 );             " Port 1 Read/Write
    PORT2_ACTIVE     = ( ADRS[0:3] = 4 T 2 );             " Port 2 Read/Write
    PORT3_ACTIVE     = ( ADRS[0:3] = 4 T 3 );             " Port 3 Read/Write
    PORT0_ANALYZE    = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16413 ) $\wedge$ BREQ*;  " 401D hex
    PORT1_ANALYZE    = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16477 ) $\wedge$ BREQ*;  " 405D hex
    PORT2_ANALYZE    = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16541 ) $\wedge$ BREQ*;  " 409D hex
    PORT3_ANALYZE    = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16605 ) $\wedge$ BREQ*;  " 40DD hex
    PORT0_RESET      = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16409 ) $\wedge$ BREQ*;  " 4019 hex
    PORT1_RESET      = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16473 ) $\wedge$ BREQ*;  " 4059 hex
    PORT2_RESET      = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16537 ) $\wedge$ BREQ*;  " 4099 hex
    PORT3_RESET      = $\overline{R/W*}$ $\wedge$ ( ADRS = 16 T 16601 ) $\wedge$ BREQ*;  " 40D9 hex
    PORT0_ERROR      = R/W* $\wedge$ ( ADRS = 16 T 16409 ) $\wedge$ BREQ*;  " 4019 hex
    PORT1_ERROR      = R/W* $\wedge$ ( ADRS = 16 T 16473 ) $\wedge$ BREQ*;  " 4059 hex
    PORT2_ERROR      = R/W* $\wedge$ ( ADRS = 16 T 16537 ) $\wedge$ BREQ*;  " 4099 hex
    PORT3_ERROR      = R/W* $\wedge$ ( ADRS = 16 T 16601 ) $\wedge$ BREQ*;  " 40D9 hex
```

Figure 10.3. Con't.

State #2 Con't.

PORT0_IN_FF_FLG	= R/W* ∧(ADRS = 16 T 16405) ∧ BREQ*;	" 4015	hex
PORT1_IN_FF_FLG	= R/W* ∧(ADRS = 16 T 16469) ∧ BREQ*;	" 4055	hex
PORT2_IN_FF_FLG	= R/W* ∧(ADRS = 16 T 16533) ∧ BREQ*;	" 4095	hex
PORT3_IN_FF_FLG	= R/W* ∧(ADRS = 16 T 16597) ∧ BREQ*;	" 40D5	hex
PORT0_IN_EF_FLG	= R/W* ∧(ADRS = 16 T 16401) ∧BREQ*;	" 4011	hex
PORT1_IN_EF_FLG	= R/W* ∧(ADRS = 16 T 16465) ∧BREQ*;	" 4051	hex
PORT2_IN_EF_FLG	= R/W* ∧(ADRS = 16 T 16529) ∧BREQ*;	" 4091	hex
PORT3_IN_EF_FLG	= R/W* ∧(ADRS = 16 T 16593) ∧BREQ*;	" 40D1	hex
PORT0_IN_HF_FLG	= R/W* ∧(ADRS = 16 T 16397) ∧ BREQ*;	" 400D	hex
PORT1_IN_HF_FLG	= R/W* ∧(ADRS = 16 T 16461) ∧ BREQ*;	" 404D	hex
PORT2_IN_HF_FLG	= R/W* ∧(ADRS = 16 T 16525) ∧ BREQ*;	" 408D	hex
PORT3_IN_HF_FLG	= R/W* ∧(ADRS = 16 T 16589) ∧ BREQ*;	" 40CD	hex
PORT0_OUT_FF_FLG	= R/W* ∧ (ADRS = 16 T 16393) ∧ BREQ*;	" 4009	hex
PORT1_OUT_FF_FLG	= R/W* ∧ (ADRS = 16 T 16457) ∧ BREQ*;	" 4049	hex
PORT2_OUT_FF_FLG	= R/W* ∧ (ADRS = 16 T 16521) ∧ BREQ*;	" 4089	hex
PORT3_OUT_FF_FLG	= R/W* ∧ (ADRS = 16 T 16585) ∧ BREQ*;	" 40C9	hex
PORT0_OUT_EF_FLG	= R/W* ∧ (ADRS = 16 T 16389) ∧ BREQ*;	" 4005	hex
PORT1_OUT_EF_FLG	= R/W* ∧ (ADRS = 16 T 16453) ∧ BREQ*;	" 4045	hex
PORT2_OUT_EF_FLG	= R/W* ∧ (ADRS = 16 T 16517) ∧ BREQ*;	" 4085	hex
PORT3_OUT_EF_FLG	= R/W* ∧ (ADRS = 16 T 16581) ∧ BREQ*;	" 40C5	hex
PORT0_OUT_HF_FLG	= R/W* ∧ (ADRS = 16 T 16385)∧ BREQ*;	" 4001	hex
PORT1_OUT_HF_FLG	= R/W* ∧ (ADRS = 16 T 16449)∧ BREQ*;	" 4041	hex
PORT2_OUT_HF_FLG	= R/W* ∧ (ADRS = 16 T 16513)∧ BREQ*;	" 4081	hex
PORT3_OUT_HF_FLG	= R/W* ∧ (ADRS = 16 T 16577)∧ BREQ*;	" 40C1	hex
BURST_BYTE_EN0	= $\overline{\text{R/W*}}$∧ (ADRS[0:7] = 8 T 15) ;	" 0F	hex
BURST_BYTE_EN1	= $\overline{\text{R/W*}}$∧ (ADRS[0:7] = 8 T 31) ;	" 1F	hex
BURST_BYTE_EN2	= $\overline{\text{R/W*}}$∧ (ADRS[0:7] = 8 T 47) ;	" 2F	hex
BURST_BYTE_EN3	= $\overline{\text{R/W*}}$∧ (ADRS[0:7] = 8 T 63) ;	" 3F	hex
STATUS0_ACTIVE	= PORT0_IN_FF_FLG ∨ PORT0_IN_EF_FLG ∨ PORT0_IN_HF_FLG ∨ PORT0_OUT_FF_FLG ∨ PORT0_OUT_EF_FLG ∨ PORT0_OUT_HF_FLG∨ PORT0_ERROR ;		
STATUS1_ACTIVE	= PORT1_IN_FF_FLG ∨ PORT1_IN_EF_FLG ∨ PORT1_IN_HF_FLG ∨ PORT1_OUT_FF_FLG ∨ PORT1_OUT_EF_FLG ∨ PORT1_OUT_HF_FLG∨ PORT1_ERROR ;		
STATUS2_ACTIVE	= PORT2_IN_FF_FLG ∨ PORT2_IN_EF_FLG ∨ PORT2_IN_HF_FLG ∨ PORT2_OUT_FF_FLG ∨ PORT2_OUT_EF_FLG ∨ PORT2_OUT_HF_FLG∨ PORT2_ERROR ;		

Figure 10.3. Con't.

```
    STATUS3_ACTIVE      = PORT3_IN_FF_FLG ∨ PORT3_IN_EF_FLG ∨
                          PORT3_IN_HF_FLG ∨ PORT3_OUT_FF_FLG ∨
                          PORT3_OUT_EF_FLG ∨ PORT3_OUT_HF_FLG∨
                          PORT3_ERROR ;
    --> ( AS*, A̅S̅*̅ ) / ( 1, 2 );  " Stay in this state till AS* goes high

END SEQUENCE

    --> ( L̅R̅E̅S̅E̅T̅*̅ ) / ( 1 );  " Go to State #1 when Reset Pulse is asserted.

  GP_RESET = PORT0_RESET ∨ PORT1_RESET∨      " Set GP Reset signal
             PORT2_RESET ∨ PORT3_RESET ;     " if any Port is to be reset.

  GP_ANALYZE = PORT0_ANALYZE ∨ PORT1_ANALYZE∨   " Set GP Analyze
               PORT2_ANALYZE ∨ PORT3_ANALYZE ;  " signal if any Port
                                                " is to be Analyzed.

END.
```

then the PORT0_ERROR signal is asserted. If the signal R/W* is low, then the signal PORT0_RESET is asserted. The AHPL description follows the memory map of the board. If a register is identified as read only and a write to that register is performed, then the Error Controller shall issue a bus error (BERR*) termination signal since no data shall be placed on the bus by the Input Port Controller. All references to the FIFO status flags and Subsystem Port (Reset, Analyze, Error) addresses listed in Table 8.1 must be single-word read or write transactions. The appropriate flag such as PORT0_IN_EF_FLG (Address = '4011' hex) shall not be set if a burst transaction is attempted by the user. The signals GP RESET and GP ANALYZE shall be used by the Error Controller to issue a synchronous termination (STERM*) when the user selects to reset or analyze a port. The Input Port Controller shall issue a synchronous termination for all other port transactions (Read, Write or Status).

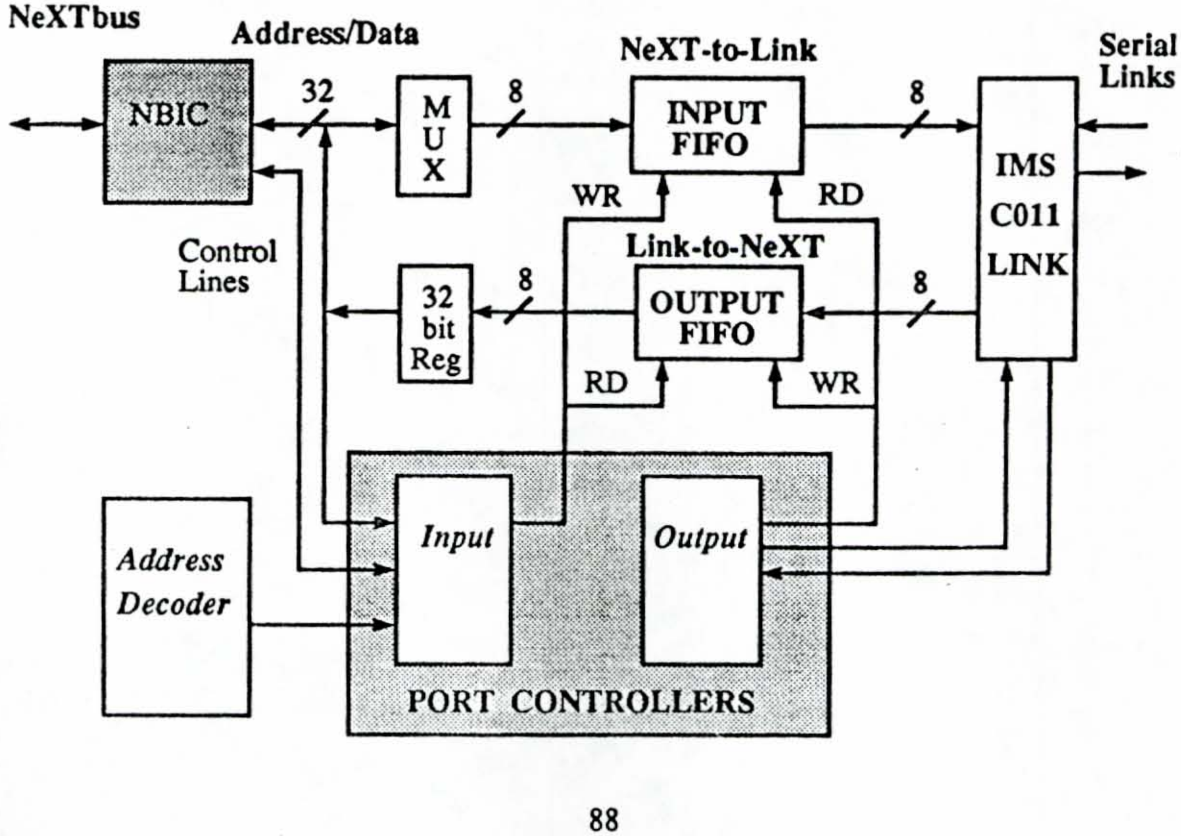
CHAPTER XI

TOP-LEVEL DESIGN OF INPUT PORT CONTROLLER

Brief Introduction to the Port Controllers

The purpose of this chapter is to describe the design of the Input Port controller. The Port Controllers are composed of an Input and Output Controller. The Input Controller is responsible for all transactions between the NBIC local bus and the Input/Output (I/O) FIFO's. The Output Controller is responsible for all transactions between the I/O FIFO's and the Inmos serial link adaptors (IMS C011). Figure 11.1 shows a block diagram of the Port Controllers interconnections with the NBIC, I/O FIFO's and the IMS C011. Both Port Controllers (input and output) work indepen-

Figure 11.1. Block Diagram of Port Controller Interconnections.



dently of each other. The multiplexor shown in Figure 11.1 is used to select the valid bytes in the 32-bit bus of the NBIC. Recall, that the local bus of the NBIC is a Big-Endian bus since the NBIC performs byte-swapping. The two SIZE bits (SIZ[1:0]) and the two least significant bits (lsb's) of the address (LAD[1:0]) determine which bytes are valid in the word. The valid transaction types are shown in Table 7.1. All bytes are shifted into the Input FIFO since the transputer instruction set is composed of bytes. The 32-bit byte loadable register is used to build a 32-bit word which shall be transferred to the NBIC. Also, the Input Port Controller is responsible for placing the appropriate FIFO status when requested by the user. The user can request status on any of the FIFO's by issuing a single-word read to the appropriate address of the board as shown in Table 8.1.

Introduction to Input/Output FIFO's

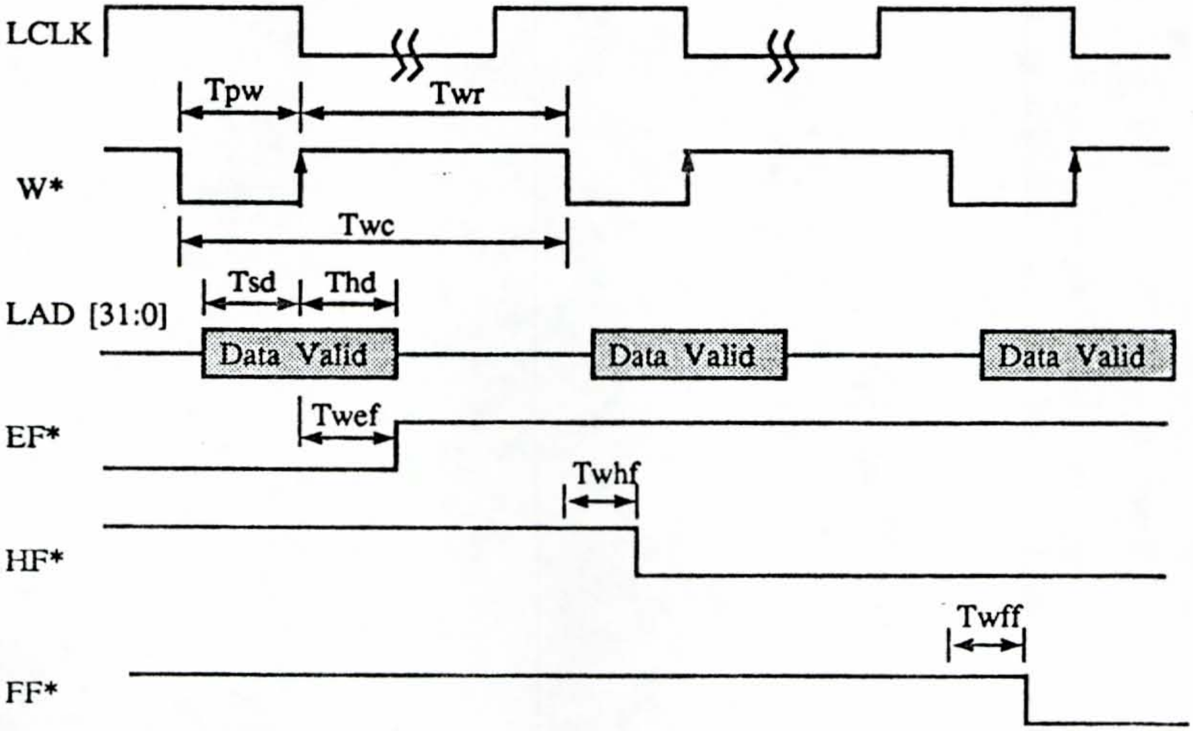
Before the design of the controllers is presented, the type of FIFO's to be used must be introduced. The reason being that the FIFO controls such write (W) and read (R) vary between FIFO's. The FIFO's selected by the author to be used in the Next-to-Motherboard interface board are the CYPRESS 74C2X-YY series. The 2X number represents the number of words the FIFO can buffer and the size of the package (300 mil or 600 mil). The YY number represents the FIFO data access time. These FIFO's contain three active low status flags which are used to determine the state of the dual port RAM. The dual port RAM refers to the architecture of the memory cell used to buffer the data. This type of architecture allows a read and write transaction to be performed independently of each other. This type of architecture is required for truly asynchronous transactions to take place between the NBIC and the array of coprocessors. The status flags available are empty FIFO (EF*), FIFO full (FF*) and FIFO half empty (HF*). A write transaction can take place if the FIFO Full flag (FF*) flag is deasserted . Similarly, a read transaction can take place if the

FIFO Empty (EF*) flag is deasserted.

These FIFO's have two types of modes known as Single Device/Width Expansion Mode or Depth Expansion Mode. Single Device/Width Expansion Mode is selected by grounding the XI input on the FIFO. During this mode, the HF* flag and retransmit features are valid. The retransmit feature (RT*) is used when transferring packets of data. It allows data to be analyzed by the receiver and can be retransmitted if necessary. This feature shall not be used in the design of the interface board. The Depth Expansion mode is used when the user needs to expand the depth of the FIFO. It is suffice to state that for our interface board, the XI and RT* inputs of the FIFO must be deasserted. The FIFO's read/write timing shall be described next so the reader understands how input/output transactions are performed with the FIFO's.

A FIFO write timing diagram is shown in Figure 11.2. The data meeting the set-up time (Tsd) before the rising edge of the write pulse and the hold time (Thd) after

Figure 11.2. FIFO Write Sequence and Status Flags.



the rising edge of the write pulse shall be written into the FIFO. The status flags relevant to the rising or trailing edge of the write pulse are shown since the input controller shall use these flags to determine if data can be written or read from the input/output FIFOs. Table 11.1 shows all the timing parameters listed in Figure 11.2 for the five different FIFO's available from CYPRESS.

Table 11.1. Timing Parameters for FIFO Write Transaction.

Parameter	7C4XX-20		7C4XX-25		7C4XX-30		7C4XX-40		7C4XX-65	
	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
Tpw	20		25		30		40		65	
Twr	10		10		10		10		15	
Twc	30		35		40		50		80	
Tsd	12		15		18		20		30	
Thd	00		00		00		00		00	
Twef		25		25		30		35		60
Twhf		30		35		40		50		80
Twff		25		25		30		35		60

NOTES: All times shown above are in nanoseconds. The information for this table was obtained from the CYPRESS SEMICONDUCTOR BiCMOS/CMOS DATA BOOK, published March 1, 1990.

A FIFO read timing diagram is shown in Figure 11.3. The data shall be valid Ta nanoseconds (access time) after the trailing edge of the read pulse (R*). The data shall be invalid Tdvr nanoseconds after the rising edge of the read pulse. The data is tri-stated when the read pulse signal is at a high (deasserted) state. Also, the state of the FIFO status flags relevant to the read pulse are shown in the timing diagram since these flags shall be used by the input controller. Table 11.2 shows all the timing parameters listed in Figure 11.3 for the five different FIFO's available from CYPRESS.

Figure 11.3. FIFO Read Sequence and Status Flags.

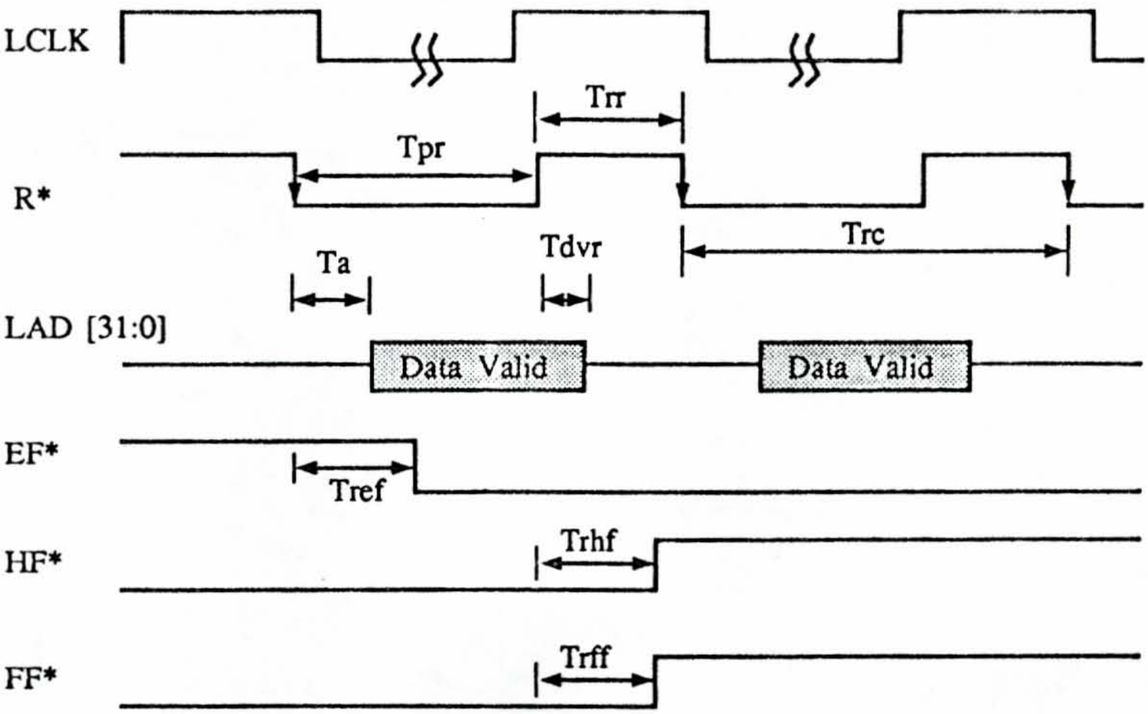


Table 11.2. Timing Parameters for FIFO Read Transaction.

Para-meter	7C4XX-20		7C4XX-25		7C4XX-30		7C4XX-40		7C4XX-65	
	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
Tpr	20		25		30		40		65	
Trr	10		10		10		10		15	
Trc	30		35		40		50		80	
Ta		20		25		30		40		65
Tdvr	03		03		03		03		03	
Tref		25		25		30		35		60
Trhf		30		35		40		50		80
Trff		25		25		30		35		60

NOTES: All times shown above are in nanoseconds. The information for this table was obtained from the CYPRESS SEMICONDUCTOR BiCMOS/CMOS DATA BOOK, published March 1, 1990.

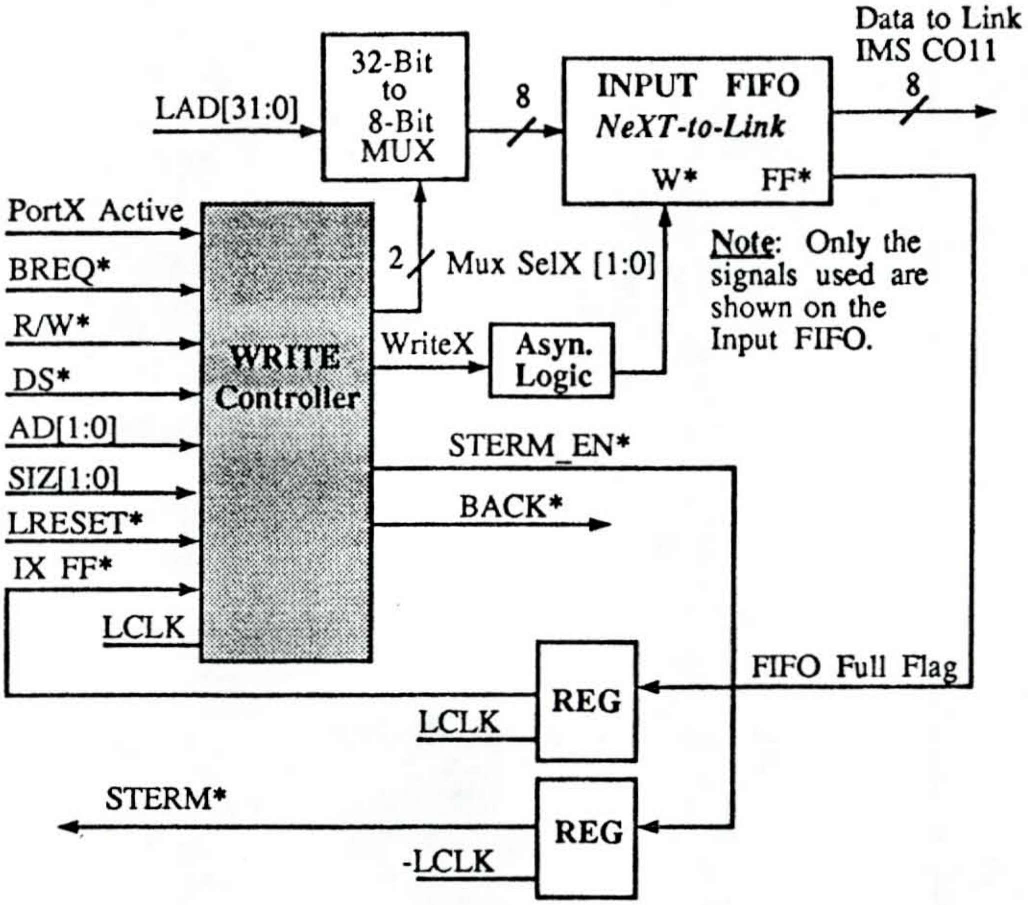
Description of Input Controller

The Input Port Controller shall consists of a Write Controller, Read Controller and Status Controller. All controllers receive control signals from the Address Decoder module and the NBIC. The Write Controller shall control all burst or single-word write transactions with the Input FIFO and shall support all valid NeXTbus transaction types. The Read Controller shall control all burst or single-word read transactions with the Output FIFO. All reads from the Output FIFO shall be 32-bit reads. Finally, the Status Controller shall place the appropriate Input/Output FIFO status flags (Half Empty, FIFO Full, FIFO Empty) or the PORT Subsystem Error Flag on the lsb's of the address/data line (LAD[0]). The first controller to be introduced is the Write Controller.

Top-level Design of Write Controller

A top-level block diagram of the Write Controller (on a per port basis) and its interconnections is shown in Figure 11.4. The Write Controller shall output the two multiplexor select signals, the write enable signal, the burst acknowledge signal and the synchronous termination signal. The asynchronous logic shown in Figure 11.4 is used to generate the appropriate write pulse (W*) to the input FIFO. The timing criteria for the write pulse is shown in Figure 11.2. The AHPL description for the Write Controller is shown in Figure 11.5. The AHPL description given in Figure 11.5 shall handle all write transaction between the NBIC and the input FIFO. Several write timing diagrams are given in Figures 11.6 through 11.10 to shown the different types of write transactions which can occur. Figure 11.6 shows a 32-bit single-word write (all four bytes are valid) transaction. Figure 11.7 shows a halfword #1 (bits [15:0]) write transaction. Figure 11.8 shows a halfword #1 write transaction with the FIFO Full Flag asserted between FIFO write pulses. The Write Controller AHPL description always checks the FIFO Full Flag before asserting the write signal which is

Figure 11.4. Top-level Block Diagram of Write Controller and its Interconnections.



used to derive the FIFO write pulse. This is also true for the synchronous termination signal (STERM_EN*). The FIFO Full Flag used in the Write AHPL description is a registered version from the Input FIFO. The FIFO's status flags are set during the cycle where a write or read transaction is being performed. The WRITE and STERM_EN* signals are connections in the AHPL during the given cycle and must remain at the same logic level; therefore, the FIFO status flags such as FIFO Full and FIFO Empty must be registered. The period of LCLK must be sufficient to allow for propagation delay of the three status flags and register setup time. Finally, Figures 11.9 and 11.10 show the two type of burst write transactions available. Figure 11.9 shows a burst write byte-mode timing diagram. During this type of transaction, only byte 0 (bits[31:24]) are valid. Figure 11.10 shows a burst write word-mode timing diagram. During this type of transaction all four bytes contain valid data.

Figure 11.5. AHPL Description for Write Controller.

```
MODULE:  WRITE_CONTROLLER
MEMORY: CNT[2], BURST*.
INPUTS:   PORTX_ACTIVE, BREQ*, R/W*,IX_FF*,  DS*;
          BURST_BYTE_ENX, AD[1:0], SIZ[1:0], LRESET*.
OUTPUTS: MUX_SELX[2], BACK*, STERM_EN*, WRITEX .

" Description of Inputs

" PORTX_ACTIVE        = Set by address decoder module when
"                       a valid transaction is to take place on Port X.
" BREQ*               = Set by NBIC when a Burst transaction
"                        is to take place (Burst transaction are 4 words).
" R/W*                = High for a Read transaction and Low for
"                       a Write Transaction.
" IX_FF*              = A registered version of the FIFO FULL status
"                       flag from the Input (NeXT-to-Link) FIFO of Port X.
" BURST_BYTE_ENX      = Set by address decoder module when burst byte
"                       transactions are to be performed on Port X.  Only
"                       byte 0 [31:24] shall be written in this mode.
" DS*                 = Data strobe signal from NBIC.
" AD[1:0]             = Two LSB of the Address from Address Decoder.
" SIZ[1:0]            = Two Size Information bits from NBIC.
" LRESET*             = Reset Pulse from NBIC (approx. 1.28 microseconds).

" Description of Outputs

" MUX_SELX[2]         = Signal used to select which byte of the 32-bit
"                       word is to be written into the Input FIFO of Port X.
" BACK*               = Signal used to acknowledge a Burst Request.
" STERM_EN*           = Synchronous termination signal set on leading
"                       edge of LCLK and clock on trailing edge of
"                       LCLK external to controller.
" WRITEX              = Signal set by controller which is used to derive
"                       the write pulse (W*) to the input FIFO of Port X.
```

BODY

1. --> ($\overline{\text{PORTX\_ACTIVE}}$ ∨ R/W*, PORTX_ACTIVE ∧ $\overline{\text{R/W*}}$) / (1, 2) ;
" Stay in State 1 if PORTX_ACTIVE is low or if there is a READ Transaction
" Go to State 2 if there is a WRITE Transaction and the PORT is active.

2. --> (DS*)/(2); " Stay in this State till data strobe (DS*) is active.
BURST* <-- BREQ* ; " Store BREQ* since it is deasserted after third Word.
CNT <-- 2 T 0;   " Clear the counter (used to keep track during Burst)

3. --> (BYTE1, (HF1 ∨ BYTE2), BYTE3) / (5, 6, 7);

4. MUX_SEL = 2 T 0 ; " Select Byte 0 [31:24]
WRITEX = IX_FF* ; " Set Write signal if FIFO full flag is deasserted.
STERM_EN* = (($\overline{\text{BYTE0}}$ ∧(BURST* ∨ $\overline{\text{BURST\_BYTE\_ENX}}$)) ∨ $\overline{\text{IX\_FF}}$*;
--> ($\overline{\text{IX\_FF*}}$, (BYTE0∨ ($\overline{\text{BURST*}}$∧BURST_BYTE_ENX)) ∧ IX_FF*) / (4,8);
BACK* <-- BREQ* ; " Assert BACK* if BREQ* is asserted

Figure 11.5. Con't.

5. MUX_SELX = 2 T 1 ; " Select Byte 1 [23:16]
WRITEX = IX_FF* ; " Set Write signal if FIFO full registered
" flag is at a High State.
STERM_EN* = $(\overline{\text{HF0}} \wedge \overline{\text{BYTE1}}) \vee \overline{\text{IX\_FF*}}$;
--> $(\overline{\text{IX\_FF*}}, (\text{HF0} \vee \text{BYTE1}) \wedge \text{IX\_FF*})$ / (5,8);

6. MUX_SELX = 2 T 2 ; " Select Byte 2 [15:8]
WRITEX = IX_FF* ; " Set Write signal if FIFO full registered
" flag is at a High State.
STERM-EN* = $(\overline{\text{BYTE2}} \vee \overline{\text{IX\_FF*}})$;
--> $(\overline{\text{IX\_FF*}}, \text{BYTE2} \wedge \text{IX\_FF*})$ / (6, 8);

7. MUX_SELX = 2 T 3 ; " Select Byte 3 [7:0]
WRITEX = IX_FF* ; " Set Write signal if FIFO full registered
" flag is at a High State.
STERM_EN* = $\overline{\text{IX\_FF*}}$;
BACK* <-- BREQ*
--> ($\overline{\text{IX\_FF*}}$) / (7);

8. WRITEX = 0; " Set WRITE and STERM_EN*
STERM_EN* = 1; " to their deasserted state.
CNT <-- INC (CNT) ; " Increment CNT which is used
--> ($\overline{\text{BURST*}} \wedge \text{CNT\_NE\_3}$) / (4); " to keep track during BURST
BACK* <-- BREQ* ; " transactions.

" If burst transaction and CNT
" not equal to three go to State #4.

9. --> (PORTX_ACTIVE, $\overline{\text{PORTX\_ACTIVE}}$) / (9, 1); " Wait in this State till
" PortX_Active signal
" is deasserted by the
" address decoder.

END SEQUENCE

WORD = $(\overline{\text{SIZ[1]}} \wedge \overline{\text{SIZ[0]}} \wedge \overline{\text{AD[1]}} \wedge \overline{\text{AD[0]}} \wedge \text{BURST*})$; "Word [31:0]
HF0 = $(\text{SIZ[1]} \wedge \overline{\text{SIZ[0]}} \wedge \overline{\text{AD[1]}} \wedge \overline{\text{AD[0]}} \wedge \text{BURST*})$; " Halfword 0 [31:16]
HF1 = $(\text{SIZ[1]} \wedge \overline{\text{SIZ[0]}} \wedge \overline{\text{AD[1]}} \wedge \overline{\text{AD[0]}} \wedge \text{BURST*})$;" Halfword 1 [15:0]
BYTE0 = $(\overline{\text{SIZ[1]}} \wedge \text{SIZ[0]} \wedge \overline{\text{AD[1]}} \wedge \overline{\text{AD[0]}} \wedge \text{BURST*})$;" Byte 0 [31:24]
BYTE1 = $(\overline{\text{SIZ[1]}} \wedge \text{SIZ[0]} \wedge \overline{\text{AD[1]}} \wedge \text{AD[0]} \wedge \text{BURST*})$;" Byte 1 [23:16]
BYTE2 = $(\text{SIZ[1]} \wedge \text{SIZ[0]} \wedge \text{AD[1]} \wedge \overline{\text{AD[0]}} \wedge \text{BURST*})$;" Byte 2 [15:8]
BYTE3 = $(\text{SIZ[1]} \wedge \text{SIZ[0]} \wedge \text{AD[1]} \wedge \text{AD[0]} \wedge \text{BURST*})$;" Byte 3 [7:0]
CNT_NE_3 = $(\overline{\text{CNT[1]}} \wedge \overline{\text{CNT[0]}}) \vee (\overline{\text{CNT[1]}} \wedge \text{CNT[0]}) \vee (\text{CNT[1]} \wedge \overline{\text{CNT[0]}})$;
--> ($\overline{\text{LRESET*}}$) / (1); " Go to State #1 when LRESET* is asserted

END

Figure 11.6. Timing Diagram for 32-bit Word Write Sequence.

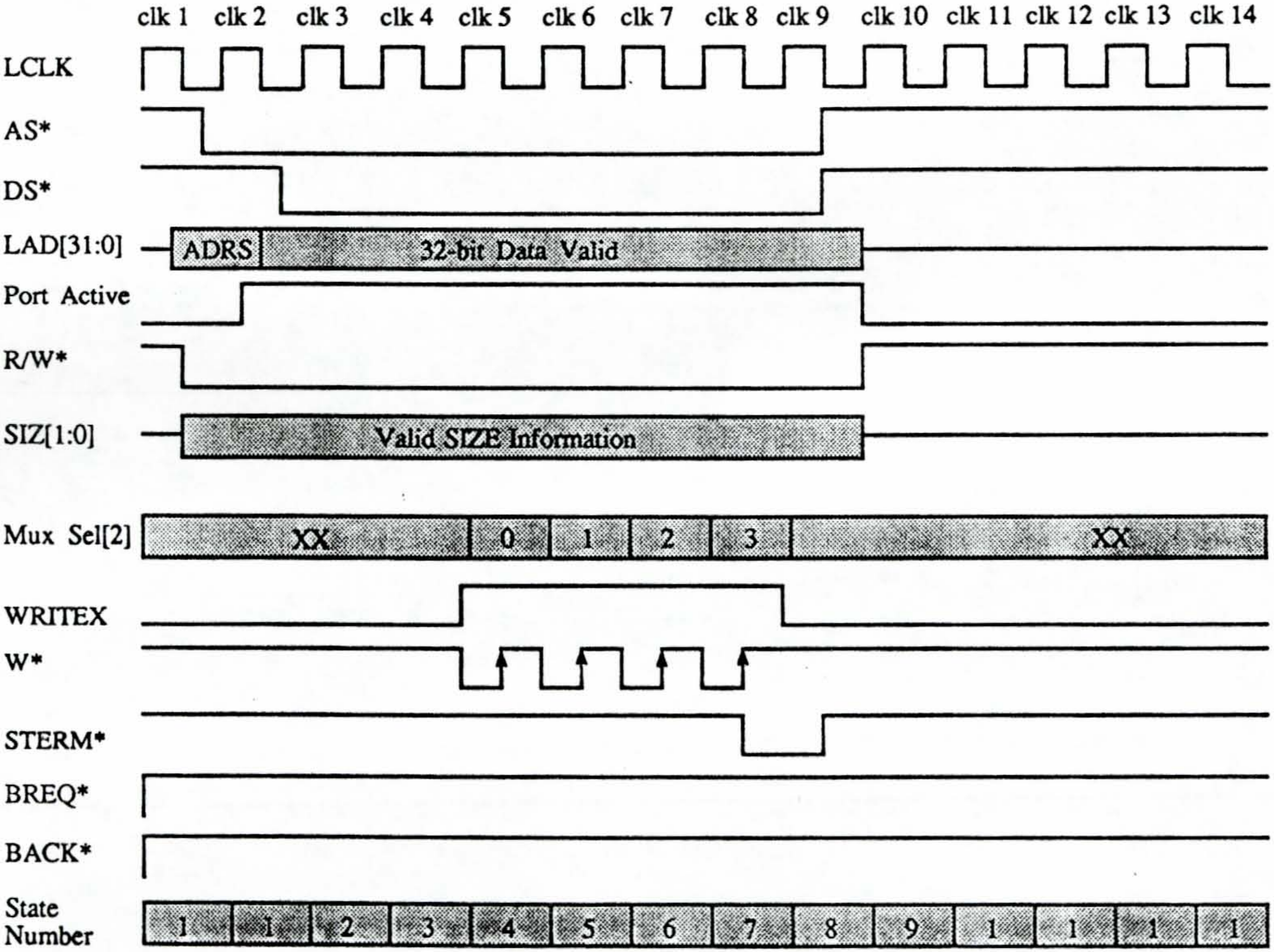


Figure 11.7. Timing Diagram for Half Word #1 Write Sequence.

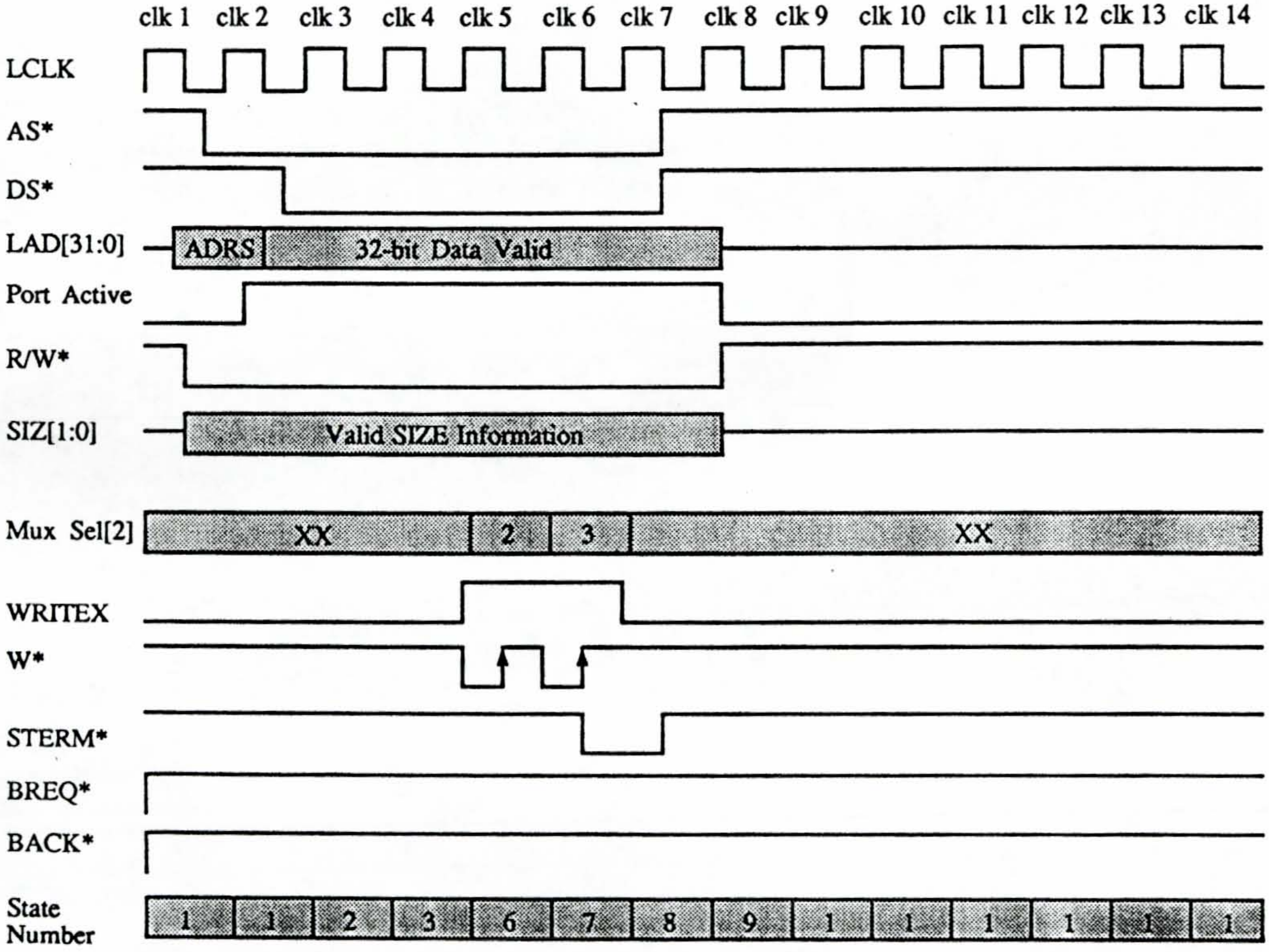


Figure 11.8. Timing Diagram for Half Word #1 Write Sequence with FIFO Full Flag Asserted.

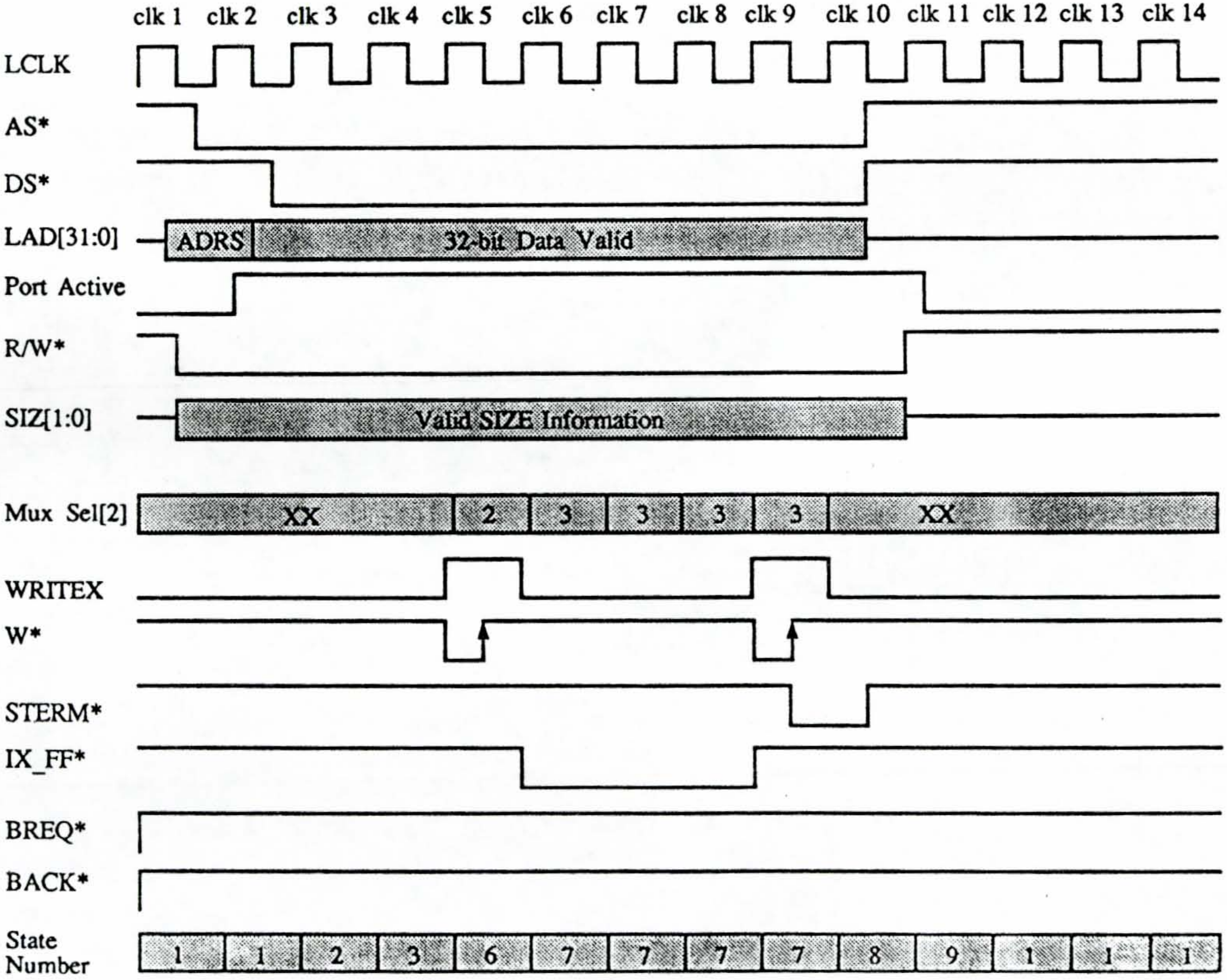


Figure 11.9. Timing Diagram for Burst (Byte Mode) Write Sequence.

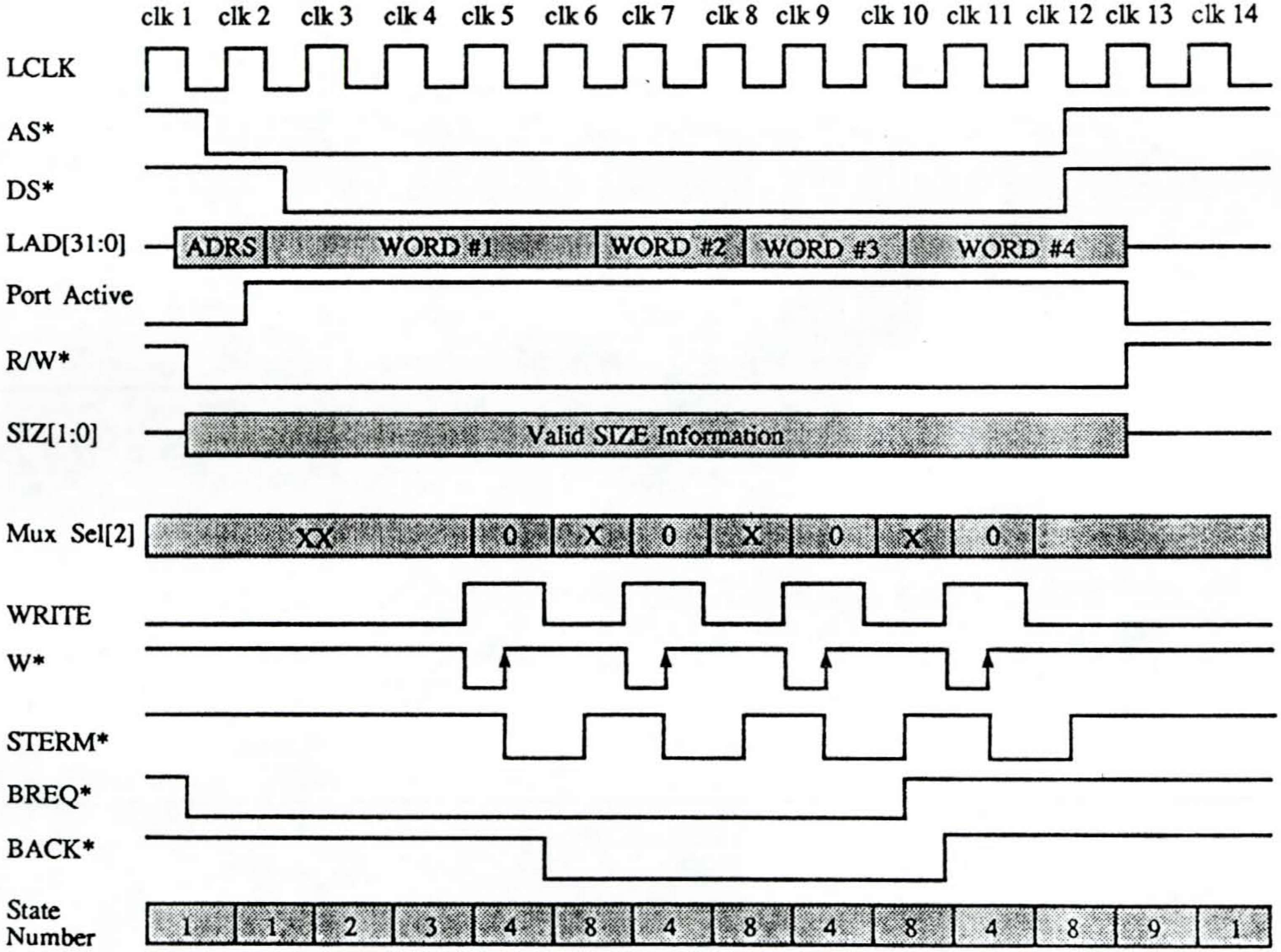


Figure 11.10. Timing Diagram for Burst (Word Mode) Write Sequence.

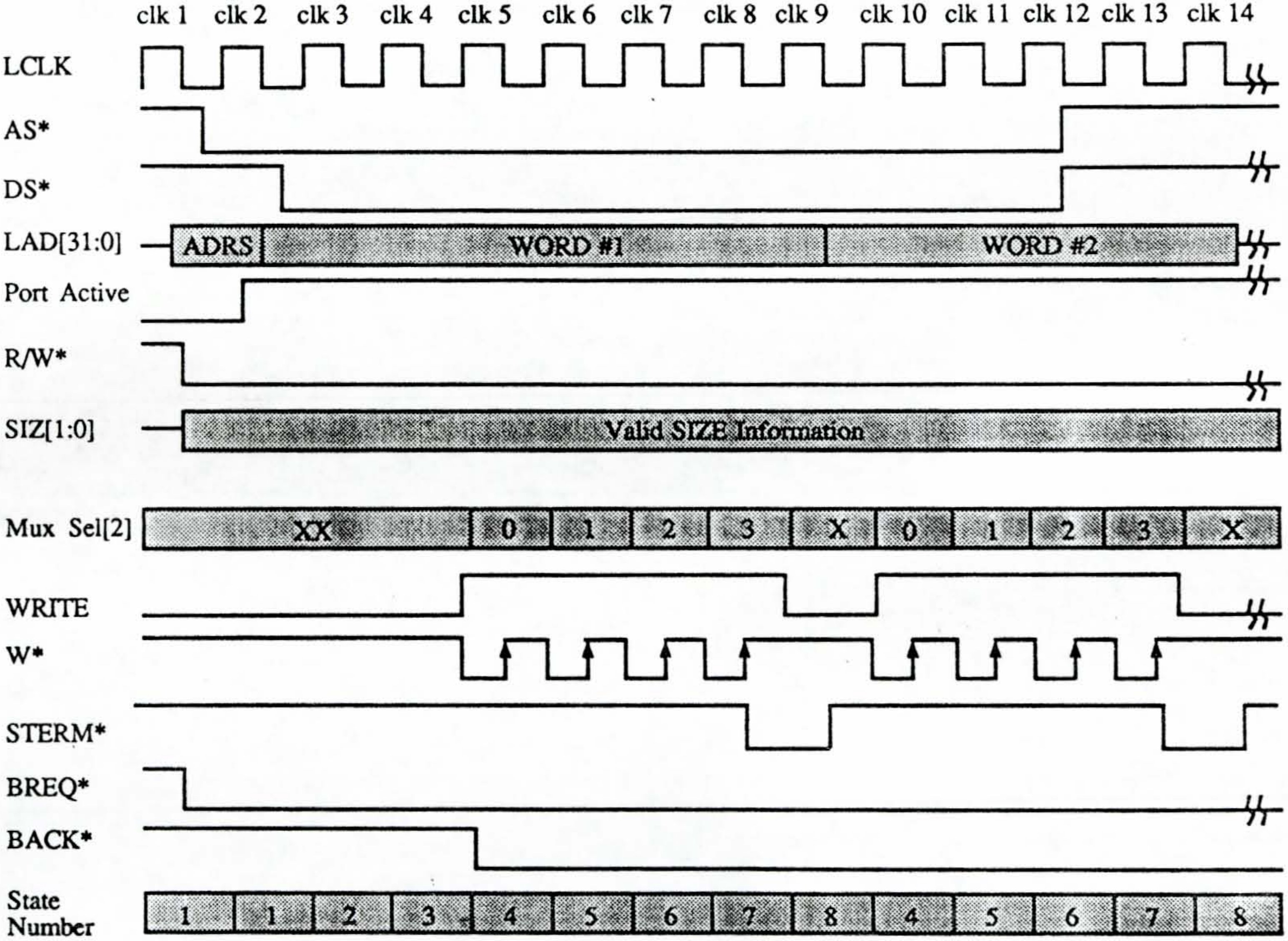
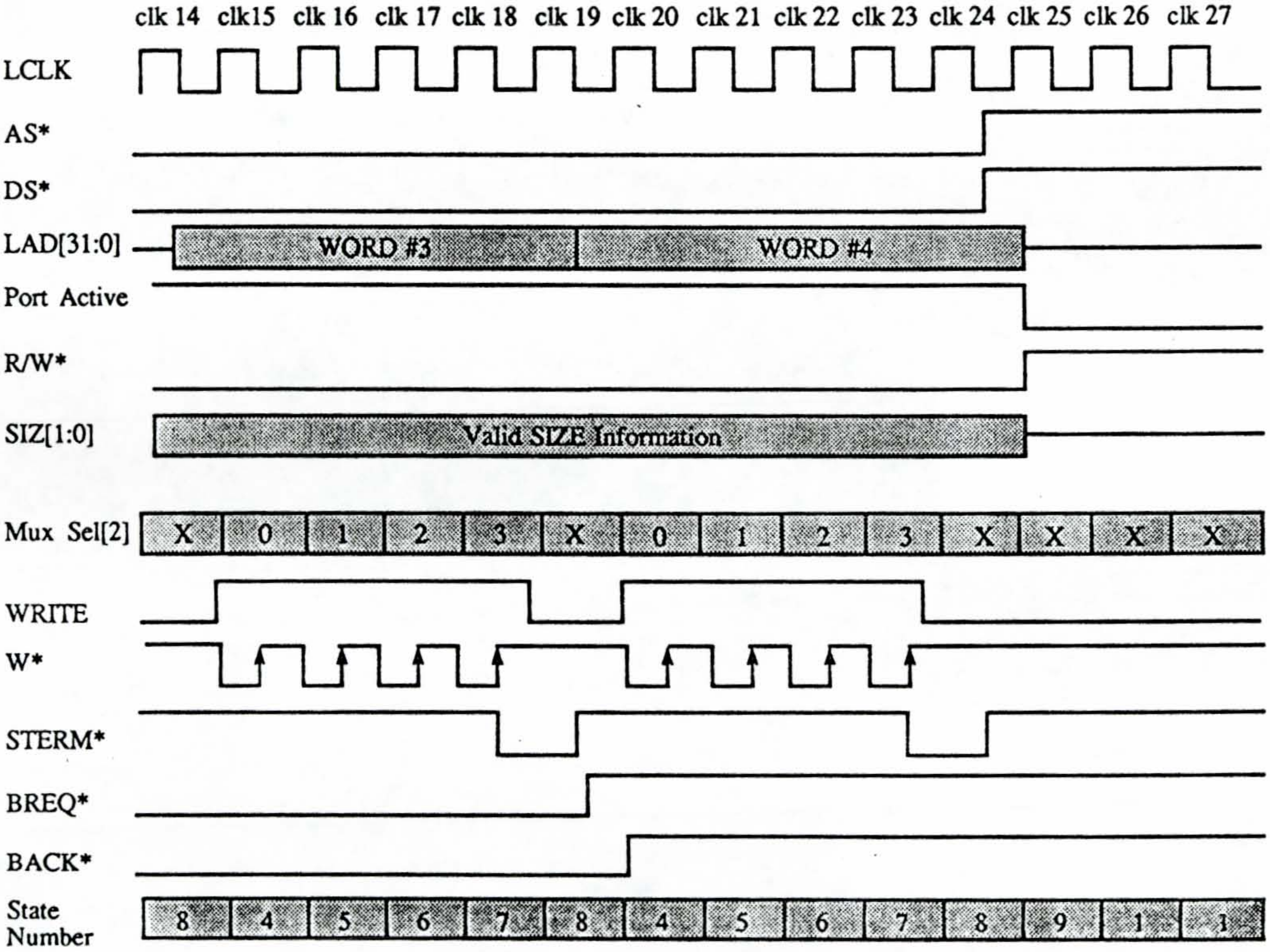


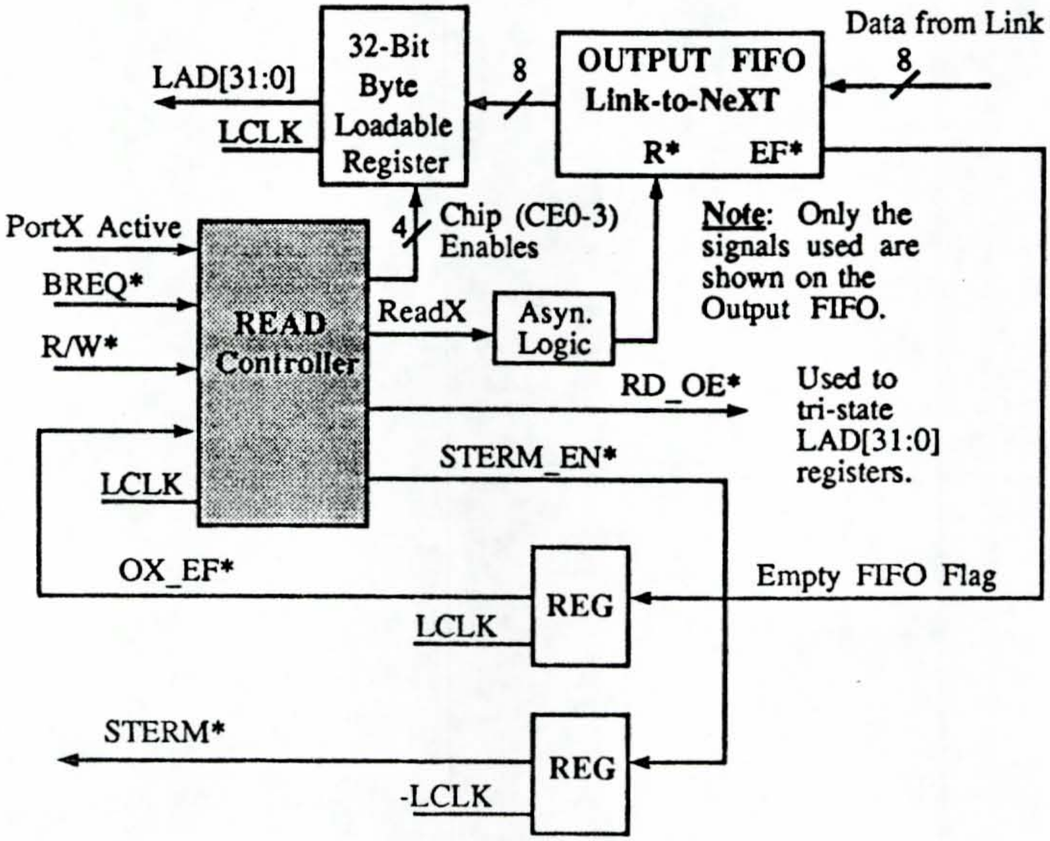
Figure 11.10. Con't.



Top-level Design of Read Controller

The Read Controller shall perform single-word and burst read transactions between the Output FIFO's and the NBIC. All read transactions shall be 32-bit word reads; therefore, four bytes must be read out of the Output FIFO. A top-level block diagram of the Read Controller (on a per port basis) and its interconnections is shown in Figure 11.11. The AHPL description for the Read Controller is shown in Figure 11.12. The 32-bit register shown in Figure 11.11 is used to selectively clock in any one of the four bytes of the 32-bit word to be transferred to the NBIC. This is accomplished by setting the chip enable (CE) of the byte to be registered. Once the four bytes have been accessed from the Output FIFO, the synchronous termination (STERM_EN*) signal shall be asserted by the Read Controller. The 32-bit byte loadable register shall also have an output enable which is controlled by the read out-

Figure 11.11. Top-level Block Diagram of Read Controller and its Interconnections.



Note: Only the signals used are shown on the Output FIFO.

Used to tri-state LAD[31:0] registers.

Figure 11.12. AHPL Description for Read Controller.

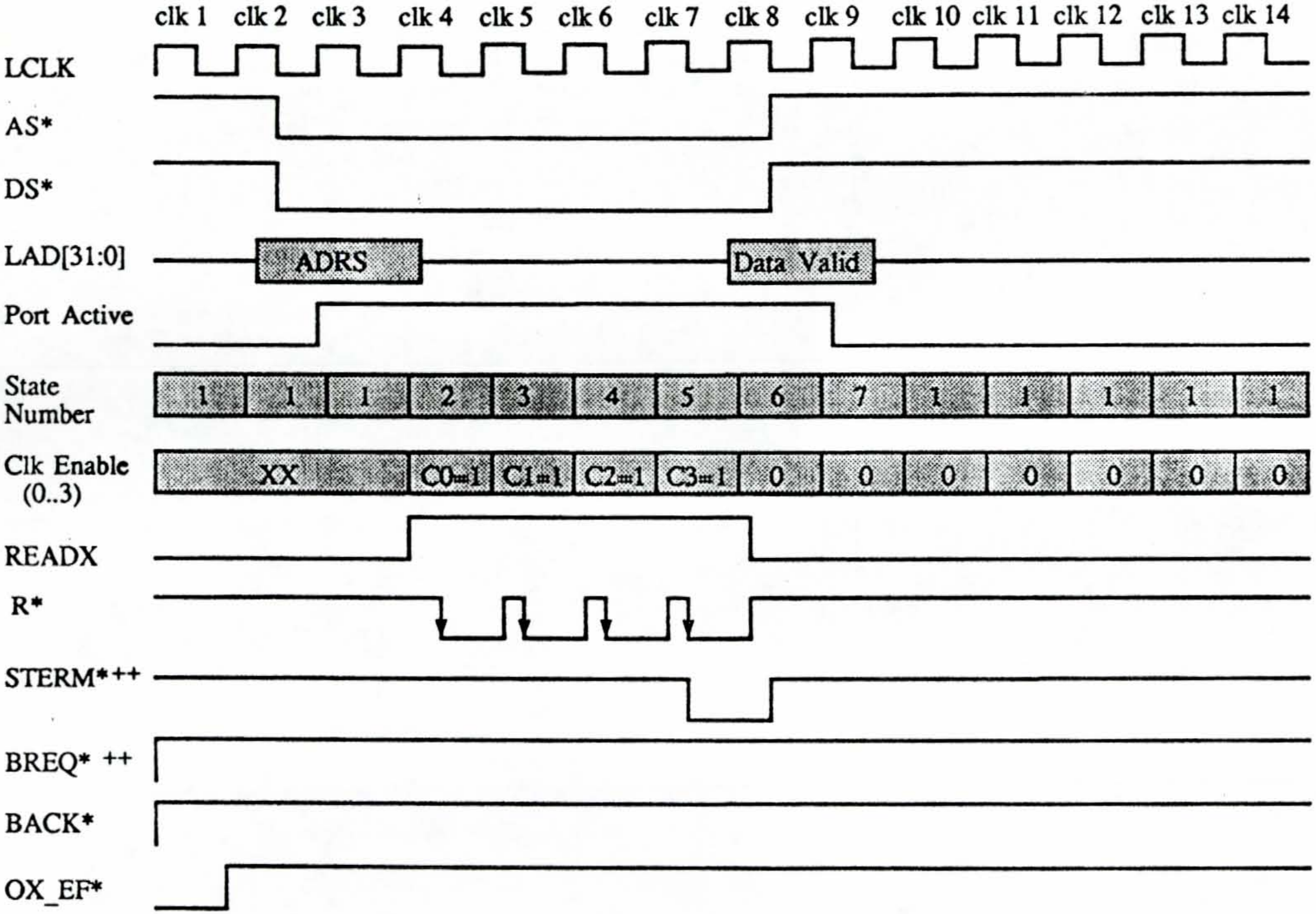
```
MODULE: READ_CONTROLLER
INPUTS:    PORTX_ACTIVE, BREQ*, R/W*, OX_EF*.
OUTPUTS: READX, CE[4], STERM_EN*, RD_OE*.
" Description of Inputs

" PORTX_ACTIVE   = Set by address decoder module when a valid
"                  Read or Write transaction is in progress on Port X.
" BREQ*          = Set by NBIC when a Burst Transaction is to
"                  take place. (All burst transactions are 4 words).
" R/W*           = High for a Read transaction and Low for a
"                  Write transaction.
" OX_EF*         = A registered version of the Empty FIFO status
"                  from the Output (Link-to-NeXT) FIFO of Port X.
" Description of Outputs

" READX          = A signal set by the controller which is used to
"                  derive the read pulse to the Output FIFO of Port X.
" CE[0:3]        = Chip enables to clock data into the 32-bit byte
"                  loadable register on a per byte basis.
" STERM_EN*      = Synchronous termination signal set on leading
"                  edge of LCLK and clock on trailing edge of
"                  LCLK external to Input Controller.
" RD_OE*.        = Output enable for 32-bit byte loadable register.
" BACK*          = Burst Acknowledge signal (Active Low) to NBIC.

BODY

1. --> ($\overline{\text{PORTX\_ACTIVE}} \vee \overline{\text{R/W*}}$, PORTX_ACTIVE $\wedge$R/W*) / ( 1, 2) ;
    " Stay in State 1 PORTX_ACTIVE is low or if there is a Write Transaction
    " Go to State 2 if there is a Read Transaction.

2. CE0      = OX_EF* ;          " Set Chip Enable 2E00 if empty FIFO flag is High
   READX = OX_EF* ;             " Set READ signal if empty FIFO flag is High
   STERM_EN* = 1 ;              " Deassert Synchronous Termination
   --> ( $\overline{\text{OX\_EF*}}$ ) / ( 2 ) ;   " Stay in this state is empty FIFO flag is LOW
   BACK* <-- BREQ* ;            " Assert BACK* if BREQ* is asserted.

3. CE1      = OX_EF* ;          " Set Chip Enable 2E01 if empty FIFO flag is High
   READX = OX_EF* ;             " Set READ signal if empty FIFO flag is High
   STERM_EN* = 1 ;              " Deassert Synchronous Termination
   --> ( $\overline{\text{OX\_EF*}}$ ) / ( 3 ) ;   " Stay in this state is empty FIFO flag is LOW

4. CE2      = OX_EF* ;          " Set Chip Enable 2E02 if empty FIFO flag is High
   READX = OX_EF* ;             " Set READ signal if empty FIFO flag is High
   STERM_EN* = 1 ;                   " Deassert Synchronous Termination
   --> ( $\overline{\text{OX\_EF*}}$ ) / ( 4 ) ;   " Stay in this state is empty FIFO flag is LOW
```

Figure 11.12. Con't.

```
5. CE3    = OX_EF* ;          " Set Chip Enable 2E03 if empty FIFO flag is High
   READX = OX_EF* ;           " Set READ signal if empty FIFO flag is High
    STERM_EN* = $\overline{OX\_EF*}$ ;  " Deassert Synchronous Termination
   --> ( $\overline{OX\_EF*}$, OX_EF* $\wedge$ $\overline{BREQ*}$) / ( 5, 2);
  " Stay in this State if EF_REG_O* is asserted.  If OX_EF* is
  " deasserted and Burst transaction go to State 2 else go to State 6

6. READX = 0;              " Word has been built so kill READ signal
    STERM_EN* = 1 ;  "Deassert Synchronous Termination

7. --> ( PORTX_ACTIVE, $\overline{PORTX\_ACTIVE}$ ) / ( 7, 1 ) ;   " Wait for PortX_Active
                                                          " to go LOW.

END SEQUENCE

RD_OE* = 0;   " While performing a READ Transaction set OE low.
--> ( $\overline{LRESET*}$) / (1).

END
```

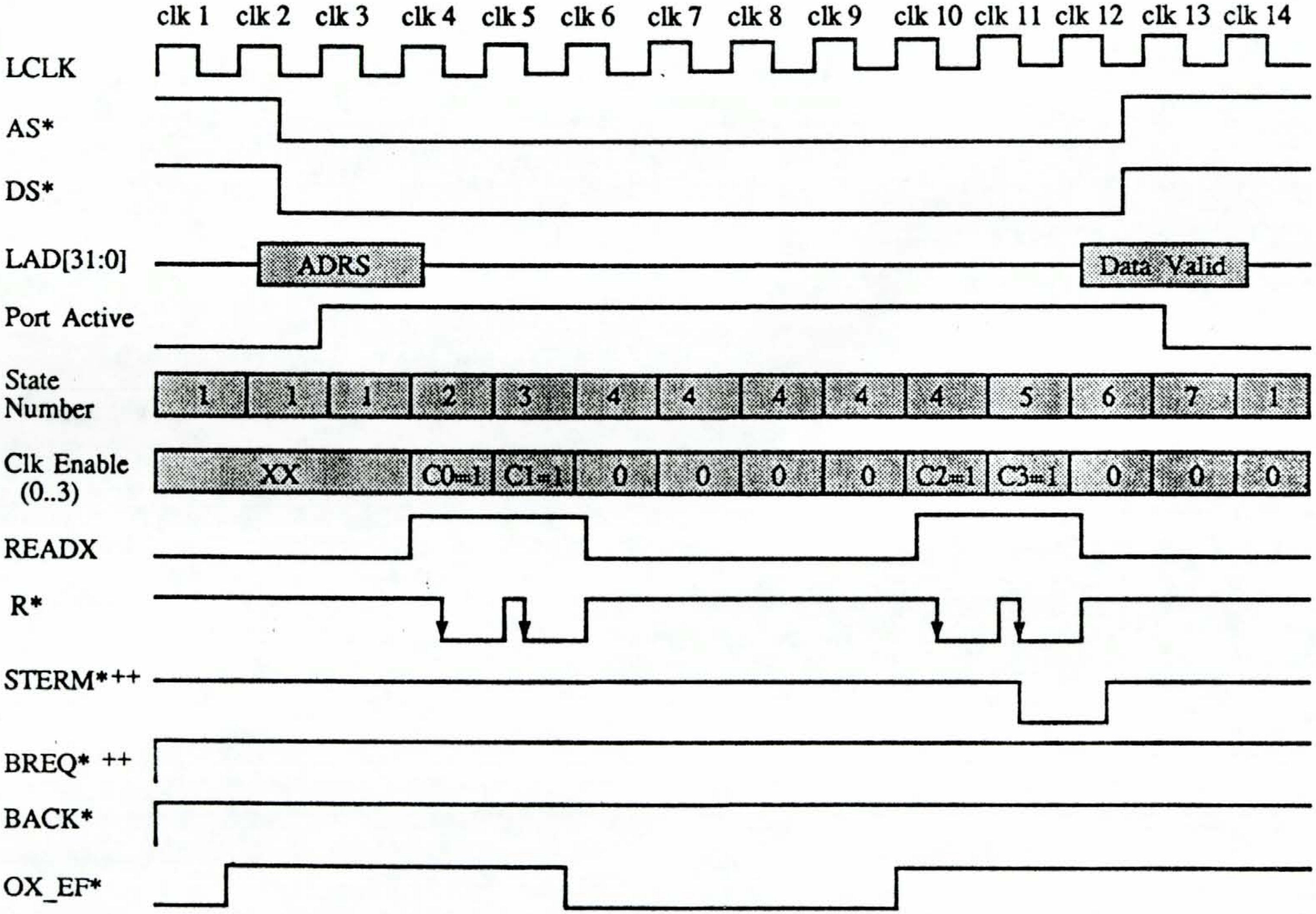
put enable (RD_OE*) signal from the Read Controller. Several timing diagrams are presented to show the sequence of states of the AHPL description. Figure 11.13 shows a single-word transaction where four bytes are accessed from the output FIFO. Figure 11.14 shows a single-word transaction where the Empty FIFO flag is asserted during the transaction. The AHPL description handles the case during any of the read states where the Output FIFO has no data to be processed. The Read Controller shall wait in that particular state until data has been entered into the Output FIFO by the Output Port Controller. Finally, Figure 11.15 shows a burst read transaction. Recall, that burst read transactions shall consists of four 32-bit words. The SIZE (SIZ[1:0]) information is not relevant during burst transactions because the NBIC only handles burst transactions of four words. This is a limitation of the NBIC. The NBIC specification does not state what would occur if a 16 or 32 word burst transaction is attempted over the NextBus. The NBIC does not provide any error checking for the size (number of words) of the burst transaction. The Read Controller AHPL description assumes all burst transactions consists of four words. The next controller to be introduced is the Status Controller.

Figure 11.13. Timing Diagram for Input Port Controller Single-Word Read Sequence.



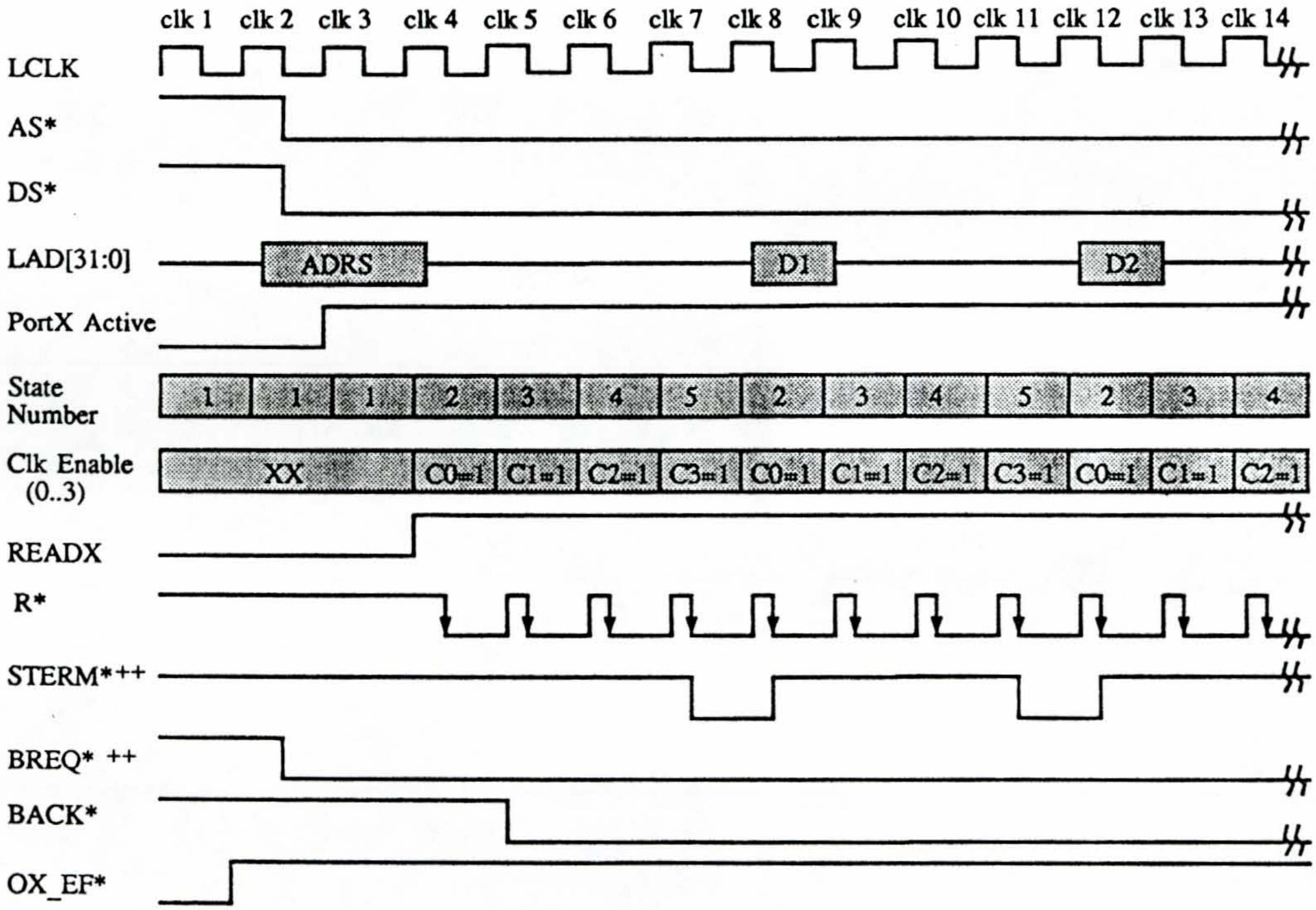
NOTES: ++ Indicates Signals asserted on the trailing edge of LCLK. (R/W* = 1).

Figure 11.14. Timing Diagram for Single-Word Read Sequence with Empty FIFO Flag Asserted.



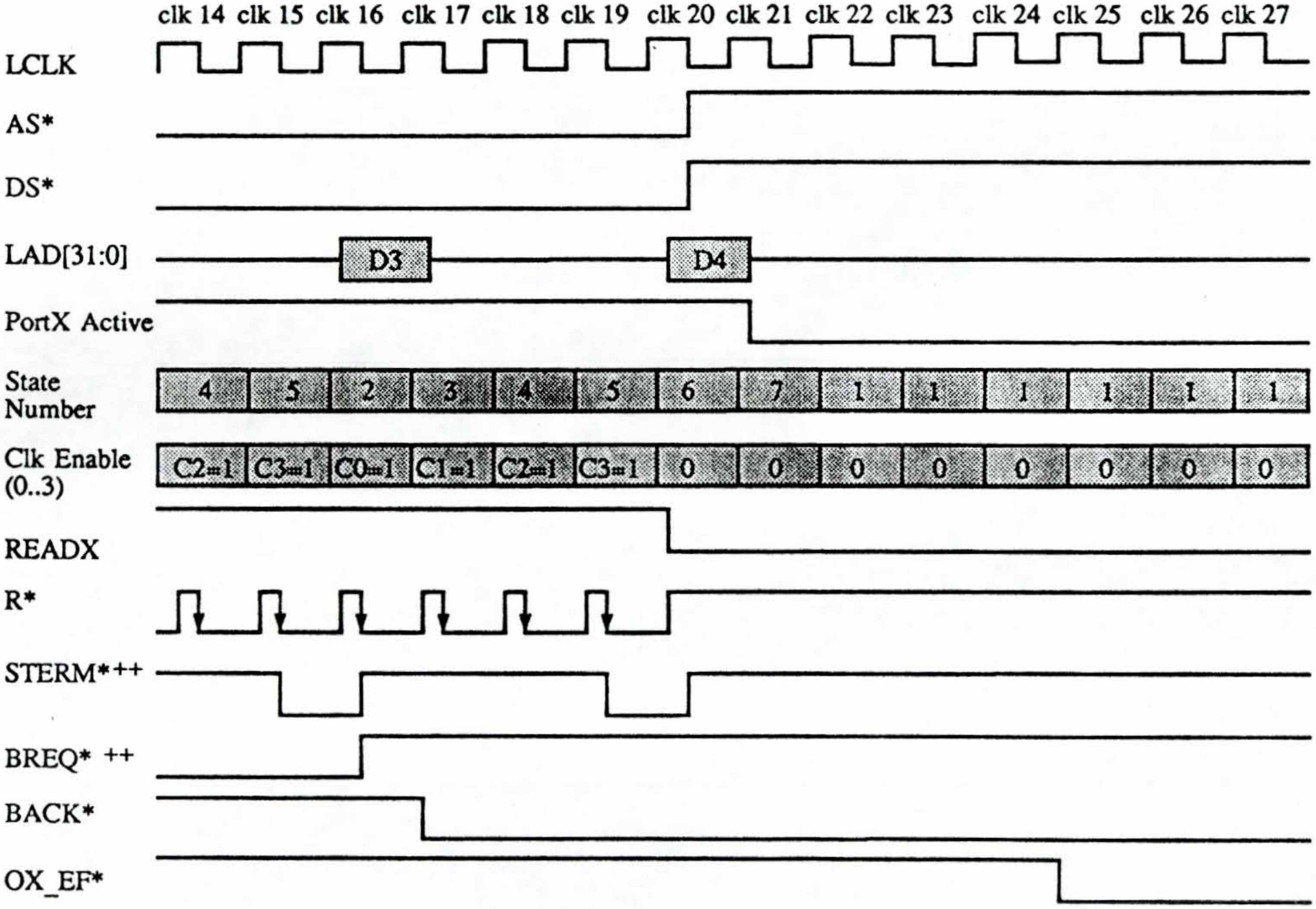
NOTES: ++ Indicates Signals asserted on the trailing edge of LCLK. (R/W* = 1).

Figure 11.15. Timing Diagram for Input Controller Burst Read Sequence.



NOTES: ++ Indicates Signals asserted on the trailing edge of LCLK. (R/W* = 1).

Figure 11.15. Con't.



NOTES: ++ Indicates Signals asserted on the trailing edge of LCLK. (R/W* = 1).

Top-level Design of Status Controller

The Status Controller shall be responsible for placing one out of seven status flags on the lsb (LAD[0]) of the address/data lines. The seven status flags are composed of six FIFO status flags and one SubSystem Error Flag (this is on a per PORT basis). The Input and Output FIFO's each have three status flags (FIFO empty, full and half empty). The Status Controller receives six FIFO registered flags along with the SubSystem Error flag. The controller shall select the appropriate flag, depending on the signal set by the Address Decoder module, and enable it on the lsb of the address/data lines. Recall, that the Read Controller has a 32-bit byte loadable register which connects to the NBIC address/data lines. This register shall be tristated while the Status Controller is active. The Status Controller shall enable its own LAD[0] data line when the user request one of the seven status flags. A top-level block diagram of the Status Controller and its interconnections is shown in Figure 11.16. The AHPL description for the Status Controller is shown in Figure 11.17. Finally, a timing diagram is shown in Figure 11.18. which shows the sequence of states executed to obtain the flag requested by the user. The reads to any of the status flags shall always be single-word transactions. The burst request (BREQ*) signal should always be deasserted when a status flag is requested. If the user request one of the seven flags by a burst read, then the Address Decoder module shall not set the appropriate flag and the transaction shall result in a Bus Error (BERR*) termination signal being asserted by the Error Controller.

Up to this point, all the controllers which make up the Input Controller have been introduced. Only one of the three controllers (Read, Write or Status) can be active at any given time. Also, the NBIC can only be involved in a transaction with one of the four ports and with one of the three controllers mentioned in this chapter. The final section of this chapter shall compile all the information for the Write, Read and Status Controller to derive the AHPL description for the Input Port Controller.

Figure 11.16. Top-level Block Diagram of Status Hardware.

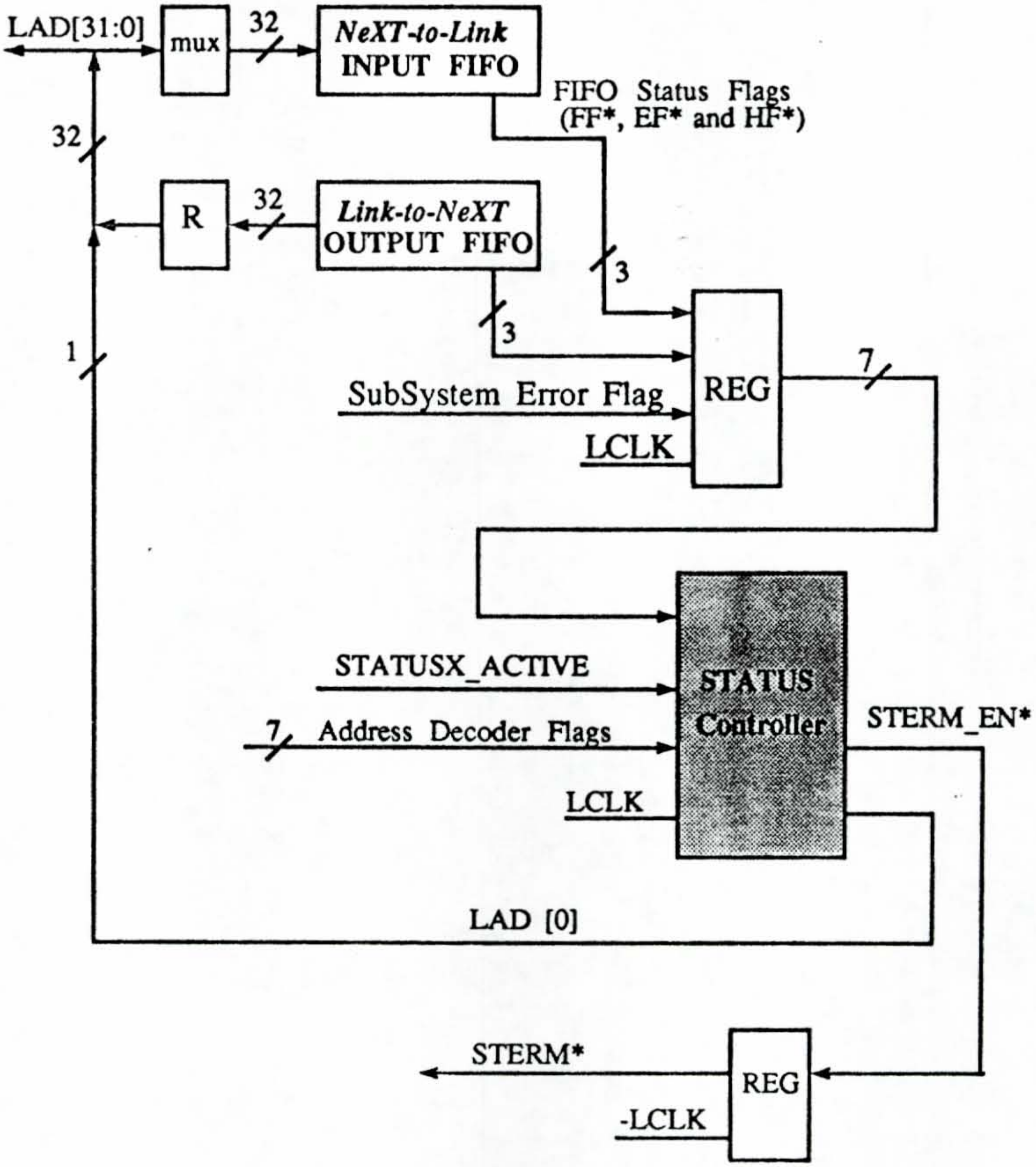


Figure 11.17. AHPL Description for Status Controller.

```
MODULE:   STATUS_CONTROLLER.
MEMORY:
INPUTS:   STATUSX_ACTIVE, PORTX_ERROR, PORTX_IN_FF_FLG,
          PORTX_IN_EF_FLG, PORTX_IN_HF_FLG, PORTX_OUT_FF_FLG,
          PORTX_OUT_EF_FLG, PORTX_OUT_HF_FLG, IX_FF*, IX_EF*,
          IX_HF*, OX_FF*, OX_EF*, OX_HF*, SUBX_ERROR, LRESET*.

OUTPUTS:  LAD[0], STERM_EN*.

" Description of Inputs

" STATUSX_ACTIVE     = Set by Address Decoder (AD) when a status flag
"                        has been requested by the user.
" PORTX_ERROR        = Asserted by AD when the Subsystem flag has been
"                        requested for PortX.
" PORTX_IN_FF_FLG    = Asserted by AD when the Input FIFO FULL flag
"                        for PortX has been requested by the user.
" PORTX_IN_EF_FLG    = Asserted by AD when the Input EMPTY FIFO
"                        flag for PortX has been requested by the user.
" PORTX_IN_HF_FLG    = Asserted by AD when the Input FIFO HALF FULL
"                        flag for PortX has been requested by the user.
" PORTX_OUT_FF_FLG = Asserted by AD when the Output FIFO FULL flag
"                        for PortX has been requested by the user.
" PORTX_OUT_EF_FLG = Asserted by AD when the Output EMPTY FIFO
"                        flag for PortX has been requested by the user.
" PORTX_OUT_HF_FLG = Asserted by AD when the Output FIFO HALF FULL
"                        flag for PortX has been requested by the user.
" IX_FF*             = PortX registered Input FIFO FULL flag.
" IX_EF*             = PortX registered Input EMPTY FIFO flag.
" IX_HF*             = PortX registered Input FIFO HALF FULL flag.
" OX_FF*             = PortX registered Output FIFO FULL flag.
" OX_EF*             = PortX registered Output EMPTY FIFO flag.
" OX_HF*             = PortX registered Output FIFO HALF FULL flag.
" SUBX_ERROR         = PortX registered SubSystem ERROR  flag.
" LRESET*            = Active Low reset pulse from NBIC (1.28 microseconds).

" Description of Outputs

" LAD[0]             = LSB of the Address/Data lines to and from NBIC.
" STERM_EN*          = Synchronous Termination Signal.  Asserted during
"                       rising edge of LCLK from Status Controller and clocked
"                       on trailing edge of LCLK external to controller.
```

BODY

1. --> ($\overline{\mathrm{STATUSX\_ACTIVE}}$) / (1).   " Stay in this State till StatusX_Active
.   is asserted by Address Decoder.

Figure 11.17. Con't.

2. STERM_EN* = 0;   " Select the appropriate Status Flag .
    LAD[0] <--   (IX_FF* ! IX_EF* ! IX_HF* ! OX_FF* ! OX_EF* ! OX_HF* !
                    SUBX_ERROR) * (PORTX_IN_FF_FLG, PORTX_IN_EF_FLG,
                    PORTX_IN_HF_FLG, PORTX_OUT_FF_FLG,
                    PORTX_OUT_EF_FLG, PORTX_OUT_HF_FLG, PORTX_ERROR).
  " The register LAD[0] is enabled during State #2 & 3. All other states Tri-stated

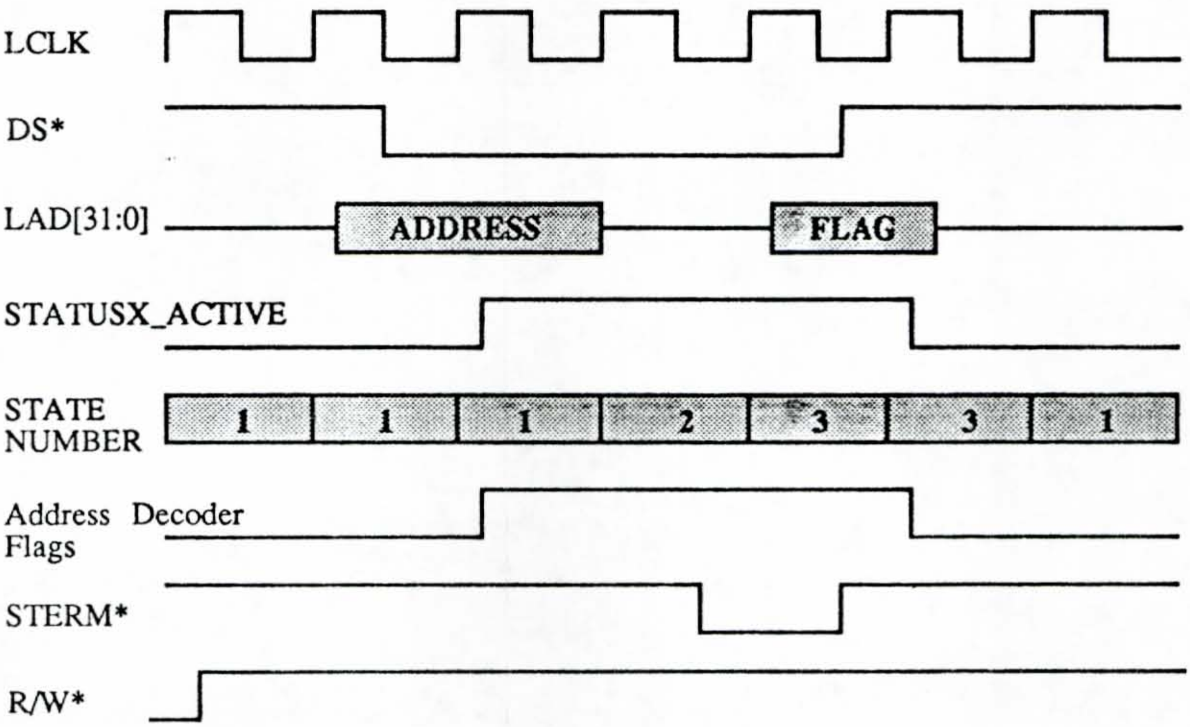
3. STERM_EN* = 1;   " Stay in this State till StatusX_Active goes low
   --> (STATUSX_ACTIVE, $\overline{\text{STATUSX\_ACTIVE}}$) / (3, 1).

END SEQUENCE

 --> (LRESET*) / (1). " Go to State #1 if Reset Pulse asserted.

END.

Figure 11.18. Timing Diagram for a Status Controller Transaction.



Input Controller Top-Level Specifications

All the controllers introduced in this chapter have been designed on a per port basis. The NBIC can only be involved in one transaction at any given time; therefore, only one of the four ports shall be active at any given time. Since only one port is active at any given time, the NeXT-to-Motherboard interface board only requires one Input Port Controller. There is no need to duplicate the same state-machine four times when it is known that only one port shall be active. Duplicating the Input Port Controller four times shall result in hardware which is only used at most one-fourth of the time. The purpose of this section is to combine the AHPL descriptions from the Write, Read and Status Controllers into a single AHPL description which shall be used in transactions involving any of the four ports. A top-level block diagram of the Input Port Controller is shown in Figure 11.19. This figure shows all 78 inputs and 18 outputs of the Input Port Controller. The symbol X represents per port signals. For example, the Port Active signal for Port 0 is denoted by PORT0_ACTIVE (X = 0 - 3).

Since there shall only be one Input Port Controller, the interface board shall contain only one 32-to-8 input multiplexor (shown in Figure 11.4) and only one 32-bit byte loadable register (shown in Figure 11.11). The output of the multiplexor shall be connected to all Input FIFO's on the interface board. Recall, that the Configuration Controller also interconnects with a 32-bit byte loadable register (shown in Figure 9.2). The designer of the board should attempt to only use one 32-bit byte loadable register. All of the Output FIFO's shall have their 8-bit data lines connected to the 32-bit byte loadable register inputs on a byte basis. Also, the data lines from the PROM in the Configuration Controller hardware shall have its 8-bit data lines connected to the same 32-bit byte loadable register. Recall, that the PROM and Output FIFO's shall have their 8-bit data lines tri-stated when not involved in a transaction. All the read transaction timing diagrams presented in this chapter (Figures 11.13 - 11.15) show the read pulse (R*) deasserted until its time to access data from the appropriate Output FIFO. If the read pulse is deasserted, then the 8-bit data lines of the FIFO are tri-stated. The designer needs to generate one global output enable for the 32-bit byte loadable register using the output enables generated by the Configuration and Input Port Controllers.

The state-diagram for the Input Port Controller is shown in Figure 11.20. This diagram is composed of the AHPL descriptions from theWrite, Read and Status Controllers previously defined. The syntax used for the transition conditions shown in Figure 11.20 are as follows: A pound sign (#) indicates a logical "OR"; An ambersign (&) indicates a logical "AND"; An exclamation mark (!) in front of a variable stands for a logical "NOT". The AHPL description for the Input Port Controller which corresponds to the state-diagram of Figure 11.20 is shown in Figure 11.21. No timing diagrams shall be presented for the Input Port Controller AHPL description since they have been presented already. The timing diagrams previously presented in this chapter apply to the Input Port Controller AHPL description shown in Figure 11.21.

This chapter has presented a top-level design of the Input Port Controller. The AHPL description presented in this chapter shall be the basis for the detail design of the Input Port Controller. The asynchronous logic which is shown in Figures 11.4 and 11.11 shall be designed by the designer. This logic depends on the period selected for the local bus clock (LCLK). The Input Port Controller AHPL descriptions outputs four READ and WRITE signals (one per port). These signals (READ and WRITE) shall be used to derive the FIFO READ (R*) and WRITE (W*) pulses.

Figure 11.19. Top-level Block Diagram of Input Controller.

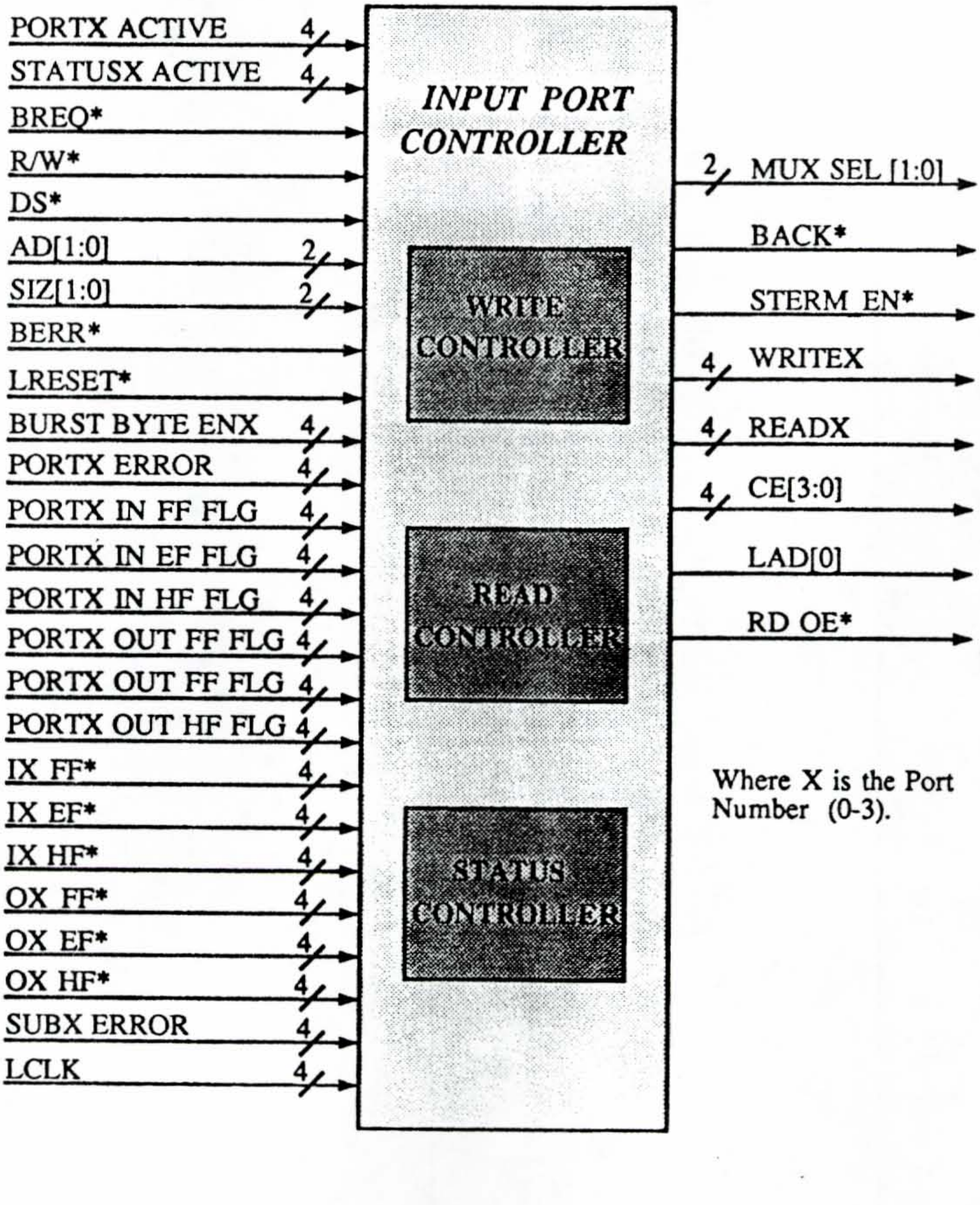


Figure 11.20. State-Diagram for Input Port Controller.

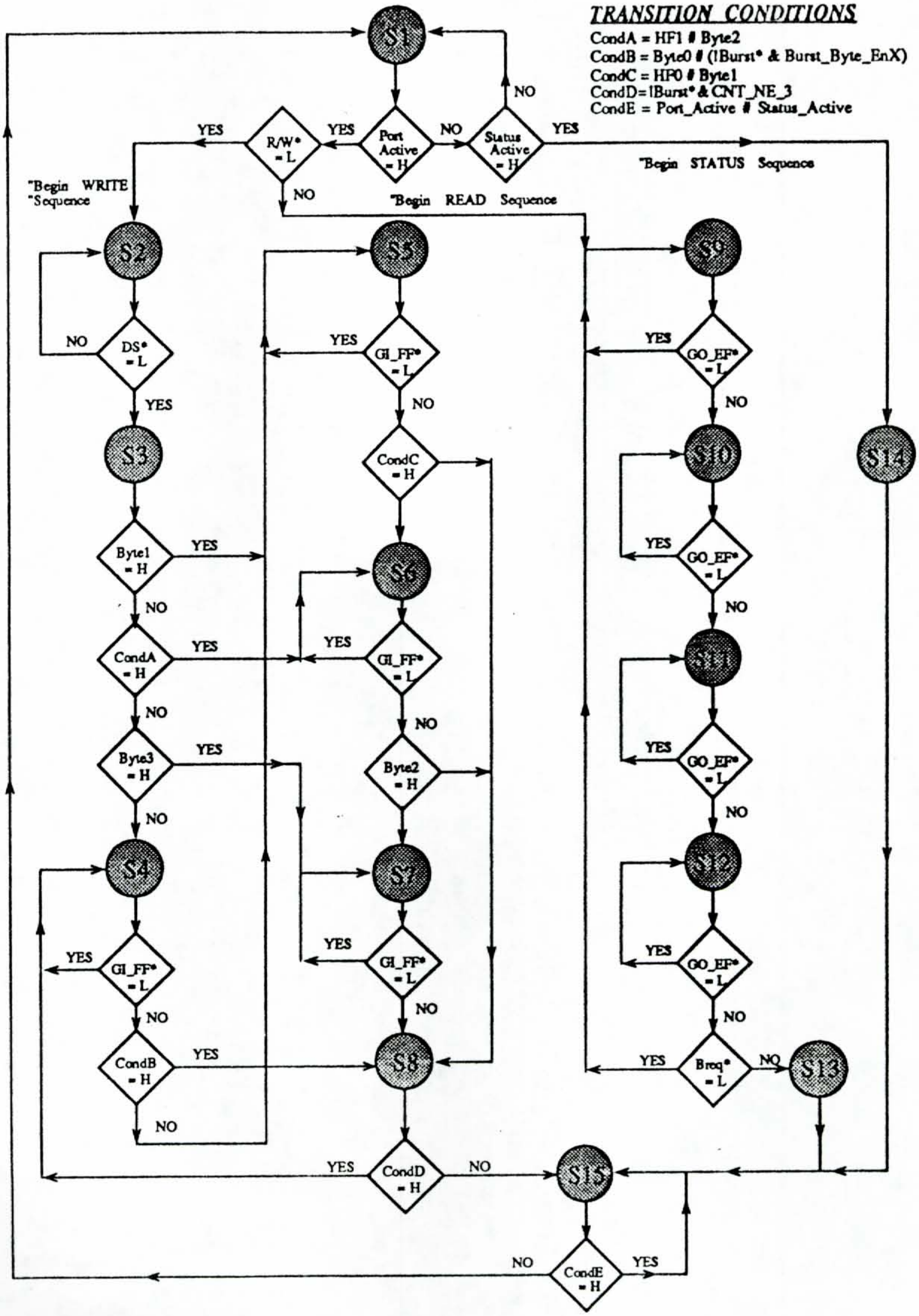


Figure 11.21. AHPL Description for Input Controller.

```
MODULE:  INPUT_CONTROLLER.
MEMORY: CNT[2], BURST*.
INPUTS:   PORT0_ACTIVE, PORT1_ACTIVE, PORT2_ACTIVE,
          PORT3_ACTIVE, STATUS0_ACTIVE, STATUS1_ACTIVE,
          STATUS2_ACTIVE, STATUS3_ACTIVE, BREQ*, R/W*,
          DS*, AD[1:0], SIZ[1:0], LRESET*, BURST_BYTE_EN0,
          BURST_BYTE_EN1, BURST_BYTE_EN2, BURST_BYTE_EN3,
          PORT0_ERROR, PORT1_ERROR, PORT2_ERROR, PORT3_ERROR,
          PORT0_IN_FF_FLG,  PORT1_IN_FF_FLG,  PORT2_IN_FF_FLG,
          PORT3_IN_FF_FLG,  PORT0_IN_EF_FLG,  PORT1_IN_EF_FLG,
          PORT2_IN_EF_FLG,  PORT3_IN_EF_FLG,  PORT0_IN_HF_FLG,
          PORT1_IN_HF_FLG,  PORT2_IN_HF_FLG,  PORT3_IN_HF_FLG,
          PORT0_OUT_FF_FLG,  PORT1_OUT_FF_FLG,  PORT2_OUT_FF_FLG,
          PORT3_OUT_FF_FLG,  PORT0_OUT_EF_FLG,  PORT1_OUT_EF_FLG,
          PORT2_OUT_EF_FLG,  PORT3_OUT_EF_FLG,  PORT0_OUT_HF_FLG,
          PORT1_OUT_HF_FLG,  PORT2_OUT_HF_FLG,  PORT3_OUT_HF_FLG,
          I0_FF*, I1_FF*, I2_FF*, I3_FF*, I0_EF*, I1_EF*, I2_EF*, I3_EF*,
          I0_HF*, I1_HF*, I2_HF*, I3_HF*, O0_FF*, O1_FF*, O2_FF*, O3_FF*,
          O0_EF*, O1_EF*, O2_EF*, O3_EF*, O0_HF*, O1_HF*, O2_HF*,
          O3_HF*, SUB0_ERR, SUB1_ERR, SUB2_ERR, SUB3_ERR, BERR*.

OUTPUTS:  MUX_SEL[2], BACK*, STERM_EN*, WRITE0, WRITE1, WRITE2,
          WRITE3, READ0, READ1, READ2, READ3, CE[4], LAD[0], RD_OE*.

" Description of Inputs
"
" PORTX_ACTIVE        = Asserted by the Address Decoder module when the
"                        user requests a Write or Read transaction with Port X.
" STATUSX_ACTIVE      = Asserted by the Address Decoder module when the
"                        user request one of the seven status flags from Port X.
" BREQ*               = Asserted by the NBIC for all burst transactions.
" R/W*                = High for a Read transaction and low for a Write.
" DS*                 = Asserted by the NBIC when data is ready on the bus
"                        (Write) or data can be received by the NBIC (Read).
"  AD[1:0]            = Two lsb's of the address latched by the Address
"                        Decoder module.
"  SIZ[1:0]           = Two size bits which are used to determine the type
"                        of transaction which is to take place.
" LRESET*             = Local bus reset signal from the NBIC.
" BURST_BYTE_ENX      = Signal(s) asserted by the Address Decoder module
"                        when the user performs a burst write (byte-mode)
"                        transaction on Port X.  Only set during burst Writes.
" PORTX_ERROR         = Asserted by the Address Decoder module when the
"                        user request the Subsystem Error flag from Port X.
"  PORTX_IN_FF_FLG    = Asserted by the Address Decoder module when the
"                        user request the Input FIFO full flag from Port X.
"  PORTX_IN_EF_FLG    = Asserted by the Address Decoder module when the
"                        user request the Input FIFO empty flag from Port X.
```

Figure 11.21. Con't.

```
" PORTX_IN_HF_FLG    = Asserted by the Address Decoder module when user
"                       request the Input FIFO half empty Flag from Port X.
" PORTX_OUT_FF_FLG   = Asserted by the Address Decoder module when user
"                       request the Output FIFO full flag from Port X.
" PORTX_OUT_EF_FLG   = Asserted by the Address Decoder module when user
"                       request the Output FIFO empty flag from Port X.
" PORTX_OUT_HF_FLG   = Asserted by the Address Decoder module when user
"                       request the Output FIFO half empty flag from Port X.
" IX_FF*              = Registered Input FIFO full flag from Port X.
" IX_EF*              = Registered Input FIFO empty flag from Port X.
" IX_HF*              = Registered Input FIFO half empty flag from Port X.
" OX_FF*              = Registered Output FIFO full flag from Port X.
" OX_EF*              = Registered Output FIFO empty flag from Port X.
" OX_HF*              = Registered Output FIFO half empty flag from Port X.
" SUBX_ERR            = Registered Subsystem Error Flag from Port X.
" BERR*               = Signal asserted by NBIC if a timeout error occurs or
"                       asserted by Error Controller if an invalid address on
"                       the board is referenced by the user.

" Description of Outputs
"
" MUX_SEL[1:0]        = Multiplexor select lines which are used during Write
"                       transactions to select appropriate byte of 32-bit word.
" BACK*               = Asserted by Input Controller to acknowledge burst
"                       request from the NBIC.
" STERM_EN*           = Asserted on rising edge of LCLK to terminate a
"                       transaction with the NBIC.  Used to generate STERM*.
" WRITEX              = Asserted when a WRITE to Input FIFO of Port X is to
"                       take place.  External asynchronous logic shall generate
"                       the write pulse signal (W*) to the FIFO of Port X.
" READX               = Asserted when a READ from Output FIFO of Port X
"                       is to take place.  External asynchronous logic shall
"                       generate the read pulse signal (R*) to FIFO of Port X.
" CE[3:0]             = Chip byte enables for the 32-bit byte loadable register.
"                       This register is used to build 32-bit word during Reads.
" LAD[0]              = Least significant bit of the local bus address/data lines.
"                       This bit is asserted by Input Port Controller when the
"                       user requests one of the seven status flags.
" RD_OE*              = Output enable signal for the 32-bit byte loadable
"                       register.  Asserted during READ transactions.
```

BODY

```
1. --> ( $\overline{\text{PORT\_ACTIVE} \vee \text{STATUS\_ACTIVE}}$,
       PORT_ACTIVE$\wedge\overline{\text{R/W*}}$, PORT_ACTIVE$\wedge$R/W*,
       STATUS_ACTIVE ) / ( 1, 2, 10, 15).   " Either stay in State #1
       MUX_SEL = 2 T 0; BACK* <-- 1;         " or go execute a Write (State 2),
       WRITE0 = 0; READ0 = 0;                " Read (State 9) or Status (State 15)
       WRITE1 = 0; READ1 = 0;                " sequence.  Set all Outputs to
       WRITE2 = 0; READ2 = 0;                " their deasserted state.
       WRITE3 = 0; READ3 = 0;                " The Symbol Z indicates Tri-state.
       CE = 4 T 0 ;  LAD[0] <-- Z;
```

Figure 11.21. Con't.

```
 " Begin WRITE Sequence
2.  --> (DS*)/(2);
    BURST* <-- BREQ* ;  " Store BREQ* since it is deasserted after third Word.
    CNT <-- 2 T 0;       " Clear the counter (used to keep track during Burst)

3.  --> (BYTE1, (HF1 ∨ BYTE2), BYTE3) / ( 5, 6, 7 );

4.  MU_SEL = 2 T 0 ;                       " Select Byte 0 [31:24]
    WRITE0 = I0_FF*∧ PORT0_ACTIVE;         " Set WRITE signal if Input FIFO
    WRITE1 = I1_FF*∧ PORT1_ACTIVE;         " full flag is deasserted. WRITE
    WRITE2 = I2_FF*∧ PORT2_ACTIVE;         " signals are asserted depending
    WRITE3 = I3_FF*∧ PORT3_ACTIVE;         " on the PORT which is active.
    STERM_EN* = (($\overline{\text{BYTE0}}$ ∧(BURST* ∨ $\overline{\text{BURST\_BYTE\_EN}}$)) ∨$\overline{\text{GI\_FF*}}$;
    --> ($\overline{\text{GI\_FF*}}$,  (BYTE0∨ ($\overline{\text{BURST*}}$∧ BURST_BYTE_EN))∧ GI_FF*) / (4,8);
    BACK* <-- BREQ* ;  " Assert BACK* if BREQ* is asserted

5.  MUX_SEL = 2 T 1 ;                      " Select Byte 1 [23:16]
    WRITE0 = I0_FF*∧ PORT0_ACTIVE;         " Set WRITE signal if Input FIFO
    WRITE1 = I1_FF*∧ PORT1_ACTIVE;         " full flag is deasserted. WRITE
    WRITE2 = I2_FF*∧ PORT2_ACTIVE;         " signals are asserted depending
    WRITE3 = I3_FF*∧ PORT3_ACTIVE;         " on the PORT which is active.
    STERM_EN* = ( $\overline{\text{HF0}}$ ∧ $\overline{\text{BYTE1}}$ )∨ $\overline{\text{GI\_FF*}}$ ;
    --> ($\overline{\text{GI\_FF*}}$, (HF0∨ BYTE1 )∧ GI_FF* ) / (5,8);

6.  MUX_SEL = 2 T 2 ;                      " Select Byte 1 [15:8]
    WRITE0 = I0_FF*∧ PORT0_ACTIVE;         " Set WRITE signal if Input FIFO
    WRITE1 = I1_FF*∧ PORT1_ACTIVE;         " full flag is deasserted. WRITE
    WRITE2 = I2_FF*∧ PORT2_ACTIVE;         " signals are asserted depending
    WRITE3 = I3_FF*∧ PORT3_ACTIVE;         " on the PORT which is active.
    STERM_EN* = ( $\overline{\text{BYTE2}}$ ∨ $\overline{\text{IF\_FF*}}$);
    --> ($\overline{\text{GI\_FF*}}$, BYTE2 ∧ GI_FF*) / ( 6, 8 );

7.  MUX_SEL = 2 T 3 ;                      " Select Byte 3 [7:0]
    WRITE0 = I0_FF*∧ PORT0_ACTIVE;         " Set WRITE signal if Input FIFO
    WRITE1 = I1_FF*∧ PORT1_ACTIVE;         " full flag is deasserted. WRITE
    WRITE2 = I2_FF*∧ PORT2_ACTIVE;         " signals are asserted depending
    WRITE3 = I3_FF*∧ PORT3_ACTIVE;         " on the PORT which is active.
    STERM_EN* =  $\overline{\text{GI\_FF*}}$;
    BACK* <-- BREQ*
    --> ( $\overline{\text{GI\_FF*}}$) / ( 7 );
```

Figure 11.21. Con;t.

```
8.  WRITE0 = 0; WRITE1 = 0;        " Set WRITE and STERM_EN*
    WRITE2 = 0; WRITE3 = 0;        " to their deasserted state.
    STERM_EN* = 1;                 " Increment CNT which keep track of
    CNT <-- INC ( CNT ) ;          " number of words during burst transfers.
    --> ( BURST*∨$\overline{\text{CNT\_NE\_3}}$, $\overline{\text{BURST*}}$∧ CNT_NE_3) / (15, 4 );
    BACK* <-- BREQ* ;

    " Begin READ Sequence
9.  CE[0]    = GO_EF* ;  " Set Chip Enable 2E00 if Output FIFO empty flag is High
    READ0 = O0_EF*∧PORT0_ACTIVE ;    " Set READ signal if Output FIFO
    READ1 = O1_EF*∧PORT1_ACTIVE ;    " FIFO empty flag is deasserted.
    READ2 = O2_EF*∧PORT2_ACTIVE ;    " READ signals are asserted depending
    READ3 = O3_EF*∧PORT3_ACTIVE ;    " on the PORT which is active.
    STERM_EN* = 1 ;          " Deassert Synchronous Termination
    --> ( $\overline{\text{GO\_EF*}}$ ) / ( 9 ) ;  " Stay in this state if the empty FIFO flag is LOW
    BACK* <-- BREQ* ;        " Assert BACK* if BREQ* is asserted.

10. CE[1]    = GO_EF* ;  " Set Chip Enable 2E01 if Output FIFO empty flag is High
    READ0 = O0_EF*∧PORT0_ACTIVE ;    " Set READ signal if Output FIFO
    READ1 = O1_EF*∧PORT1_ACTIVE ;    " FIFO empty flag is deasserted.
    READ2 = O2_EF*∧PORT2_ACTIVE ;    " READ signals are asserted depending
    READ3 = O3_EF*∧PORT3_ACTIVE ;    " on the PORT which is active.
    STERM_EN* = 1 ;          " Deassert Synchronous Termination
    --> ( $\overline{\text{GO\_EF*}}$ ) / ( 10 ) ;  " Stay in this state if the empty FIFO flag is LOW

11. CE[2]    = GO_EF* ;  " Set Chip Enable 2E02 if Output FIFO empty flag is High
    READ0 = O0_EF*∧PORT0_ACTIVE ;    " Set READ signal if Output FIFO
    READ1 = O1_EF*∧PORT1_ACTIVE ;    " FIFO empty flag is deasserted.
    READ2 = O2_EF*∧PORT2_ACTIVE ;    " READ signals are asserted depending
    READ3 = O3_EF*∧PORT3_ACTIVE ;    " on the PORT which is active.
    STERM_EN* = 1 ;          " Deassert Synchronous Termination
    --> ( $\overline{\text{GO\_EF*}}$ ) / ( 11 ) ;  " Stay in this state if the empty FIFO flag is LOW

12. CE[3]    = GO_EF* ;  " Set Chip Enable 2E03 if Output FIFO empty flag is High
    READ0 = O0_EF*∧PORT0_ACTIVE ;    " Set READ signal if Output FIFO
    READ1 = O1_EF*∧PORT1_ACTIVE ;    " FIFO empty flag is deasserted.
    READ2 = O2_EF*∧PORT2_ACTIVE ;    " READ signals are asserted depending
    READ3 = O3_EF*∧PORT3_ACTIVE ;    " on the PORT which is active.
    STERM_EN* = $\overline{\text{GO\_EF*}}$ ;  " Deassert Synchronous Termination
    --> ( $\overline{\text{GO\_EF*}}$,  GO_EF* ∧ $\overline{\text{BREQ*}}$) / ( 12, 9);
    " Stay in this State if GO_EF* signal is asserted.  If GO_EF* is
    " deasserted and Burst transaction, then go to State #2 else go to State #6.
```

Figure 11.21. Con't.

```
13. READ0 = 0; READ1 = 0;   " Deassert the FIFO Read signals
    READ2 = 0; READ3 = 0;
    STERM_EN* = 1 ;          " Deassert Synchronous Termination
    --> ( 15 ).  " Go to State #15 to Wait for Port_Active to go LOW.

    " Begin STATUS Sequence
14. STERM_EN* = 0;       " Select the appropriate Status Flag .
    LAD[0] <--   ( IF_FF* ! IF_EF* ! IF_HF* ! OF_FF* ! OF_EF* ! OF_HF* !
                   SUB_ERROR ) * ( PORT_IN_FF_FLG, PORT_IN_EF_FLG,
                   PORT_IN_HF_FLG, PORT_OUT_FF_FLG,
                   PORT_OUT_EF_FLG, PORT_OUT_HF_FLG, PORT_ERROR).
   " The register LAD[0] is enabled during State #14 & 15.
   " All other states, the register is tri-stated

15. --> ( PORT_ACTIVE ∨ STATUS_ACTIVE,                                          " Wait in this State
          $\overline{\mathrm{PORT\_ACTIVE}}$ ∨ $\overline{\mathrm{STATUS\_ACTIVE}}$ ) / ( 15, 1).     " till PortX_Active
                                                                                " or StatusX_Active
                                                                                " are deasserted by
                                                                                " Address Decoder.

END SEQUENCE

WORD   = ( $\overline{\mathrm{SIZ[1]}}$ ∧ $\overline{\mathrm{SIZ[0]}}$ ∧ $\overline{\mathrm{AD[1]}}$ ∧ $\overline{\mathrm{AD[0]}}$ ∧ BURST* ); "Word [31:0]
HF0    = ( SIZ[1] ∧ $\overline{\mathrm{SIZ[0]}}$ ∧ $\overline{\mathrm{AD[1]}}$ ∧ $\overline{\mathrm{AD[0]}}$ ∧ BURST* ); " Halfword 0 [31:16]
HF1    = ( SIZ[1] ∧ $\overline{\mathrm{SIZ[0]}}$ ∧ $\overline{\mathrm{AD[1]}}$ ∧ $\overline{\mathrm{AD[0]}}$ ∧ BURST* ); " Halfword 1 [15:0]
BYTE0  = ( $\overline{\mathrm{SIZ[1]}}$ ∧ SIZ[0] ∧ $\overline{\mathrm{AD[1]}}$ ∧ $\overline{\mathrm{AD[0]}}$ ∧ BURST* ); " Byte 0 [31:24]
BYTE1  = ( $\overline{\mathrm{SIZ[1]}}$ ∧ SIZ[0] ∧ $\overline{\mathrm{AD[1]}}$ ∧ AD[0] ∧ BURST* ); " Byte 1 [23:16]
BYTE2  = ( SIZ[1] ∧ SIZ[0] ∧ AD[1] ∧ $\overline{\mathrm{AD[0]}}$ ∧ BURST* ); " Byte 2 [15:8]
BYTE3  = ( SIZ[1] ∧ SIZ[0] ∧ AD[1] ∧ AD[0] ∧ BURST* ); " Byte 3 [7:0]
CNT_NE_3 = ( $\overline{\mathrm{CNT[1]}}$ ∧ $\overline{\mathrm{CNT[0]}}$) ∨ ($\overline{\mathrm{CNT[1]}}$ ∧ CNT[0]) ∨ (CNT[1] ∧ $\overline{\mathrm{CNT[0]}}$);
--> ( $\overline{\mathrm{LRESET*}}$ ∨ $\overline{\mathrm{BERR*}}$) / ( 1 ); " Go to State #1 when LRESET* is asserted
RD_OE* = $\overline{\mathrm{PORT\_ACTIVE}}$ ∨ $\overline{\mathrm{R/W*}}$ ;

PORT_ACTIVE = PORT0_ACTIVE ∨ PORT1_ACTIVE ∨
              PORT2_ACTIVE ∨ PORT3_ACTIVE;

STATUS_ACTIVE = STATUS0_ACTIVE ∨ STATUS1_ACTIVE ∨
                STATUS2_ACTIVE ∨ STATUS3_ACTIVE;

BURST_BYTE_EN = ( BURST_BYTE_EN0 ! BURST_BYTE_EN1 !
                  BURST_BYTE_EN2 ! BURST_BYTE_EN3 ) *
                ( PORT0_ACTIVE, PORT1_ACTIVE,
                  PORT2_ACTIVE, PORT3_ACTIVE );
```

Figure 11.21. Con't.

```
PORT_ERROR = PORT0_ERROR ∨ PORT1_ERROR ∨
                  PORT2_ERROR ∨ PORT3_ERROR;
SUB_ERR = ( SUB0_ERR ! SUB1_ERR ! SUB2_ERR ! SUB3_ERR ) *
            ( STATUS0_ACTIVE, STATUS1_ACTIVE,
              STATUS2_ACTIVE, STATUS3_ACTIVE );
PORT_IN_FF_FLG = ( PORT0_IN_FF_FLG ∨ PORT1_IN_FF_FLG∨
                           PORT2_IN_FF_FLG ∨ PORT3_IN_FF_FLG );
PORT_IN_EF_FLG = ( PORT0_IN_EF_FLG ∨ PORT1_IN_EF_FLG∨
                           PORT2_IN_EF_FLG ∨ PORT3_IN_EF_FLG );
PORT_IN_HF_FLG = ( PORT0_IN_HF_FLG ∨ PORT1_IN_HF_FLG∨
                           PORT2_IN_HF_FLG ∨ PORT3_IN_HF_FLG );
PORT_OUT_FF_FLG = ( PORT0_OUT_FF_FLG ∨ PORT1_OUT_FF_FLG∨
                              PORT2_OUT_FF_FLG ∨ PORT3_OUT_FF_FLG );
PORT_OUT_EF_FLG = ( PORT0_OUT_EF_FLG ∨ PORT1_OUT_EF_FLG∨
                              PORT2_OUT_EF_FLG ∨ PORT3_OUT_EF_FLG );
PORT_OUT_HF_FLG = ( PORT0_OUT_HF_FLG ∨ PORT1_OUT_HF_FLG ∨
                              PORT2_OUT_HF_FLG∨ PORT3_OUT_HF_FLG );
IF_FF* = ( I0_FF* ! I1_FF* ! I2_FF* ! I3_FF* ) * ( STATUS0_ACTIVE,
            STATUS1_ACTIVE, STATUS2_ACTIVE, STATUS3_ACTIVE );
IF_EF* = ( I0_EF* ! I1_EF* ! I2_EF* ! I3_EF* ) * ( STATUS0_ACTIVE,
            STATUS1_ACTIVE, STATUS2_ACTIVE, STATUS3_ACTIVE );
IF_HF* = ( I0_HF* ! I1_HF* ! I2_HF* ! I3_HF* ) * ( STATUS0_ACTIVE,
            STATUS1_ACTIVE, STATUS2_ACTIVE, STATUS3_ACTIVE );
OF_FF* = ( O0_FF* ! O1_FF* ! O2_FF* ! O3_FF* ) * ( STATUS0_ACTIVE,
             STATUS1_ACTIVE, STATUS2_ACTIVE, STATUS3_ACTIVE );
OF_EF* = ( O0_EF* ! O1_EF* ! O2_EF* ! O3_EF* ) * ( STATUS0_ACTIVE,
             STATUS1_ACTIVE, STATUS2_ACTIVE, STATUS3_ACTIVE );
OF_HF* = ( O0_HF* ! O1_HF* ! O2_HF* ! O3_HF* ) * ( STATUS0_ACTIVE,
             STATUS1_ACTIVE, STATUS2_ACTIVE, STATUS3_ACTIVE );
GI_FF* = ( I0_FF* ! I1_FF* ! I2_FF* ! I3_FF* ) * ( PORT0_ACTIVE,
            PORT1_ACTIVE, PORT2_ACTIVE, PORT3_ACTIVE );
GO_EF* = ( O0_EF* ! O1_EF* ! O2_EF* ! O3_EF* ) * ( PORT0_ACTIVE,
            PORT1_ACTIVE, PORT2_ACTIVE, PORT3_ACTIVE );

END.
```

CHAPTER XII

TOP-LEVEL DESIGN OF OUTPUT PORT CONTROLLER

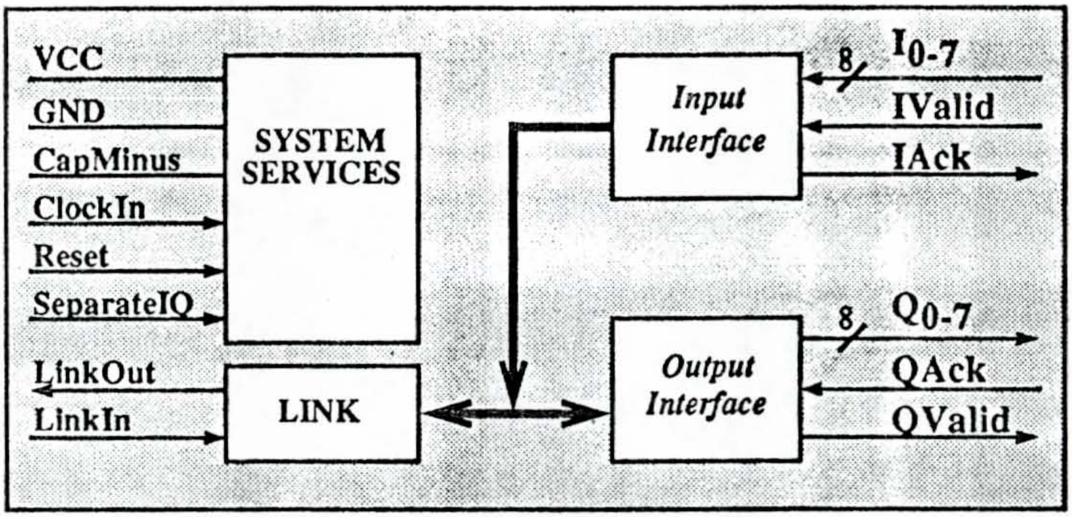
Description of the Output Port Controller

The purpose of this chapter is to present a top-level design of the Output Port Controller. This controller shall communicate with the Input/Output FIFO's and the Inmos Link Adaptor (IMS C011). The controller shall write bytes received from Link to the appropriate Output FIFO and read data from the Input FIFO so that it can be transmitted to Link. The controller shall control all the handshaking signals between the CYPRESS FIFO's and the Inmos Link Adaptor. Before the design of the Output Controller is presented, the Inmos Link Adaptor must be introduced since it interfaces with the Output Controller.

Introduction to the Inmos Link Adaptor (IMS CO11)

The Inmos Link Adaptor provides full duplex communication between a peripheral or microprocessor and the transputer serial links. The IMS C011 has two modes of operation: peripheral interface and bus interface. When the IMS C011 is configured has a peripheral interface (Mode 1), it has two byte-wide port interfaces to communicate with the serial links of the transputer. Both ports (input and output) have their own two-wire set of handshaking lines. When configured has a bus interface (Mode 2), it is used to interface an Inmos transputer serial link with a microprocessor system bus. Only the peripheral interface mode shall be used on the NeXT-to-Motherboard interface board; therefore, the bus interface mode shall not be presented. A block diagram of the IMS C011 configured in Mode 1 is shown in Figure 12.1. The system services are used to startup and maintain the link adaptor. A brief description of the system services and port interface signals are shown in Table 12.1. The designer should read the specifications of the IMS C011 link adaptor for detail design

Figure 12.1. Block Diagram of IMS C011 Configured in Mode 1.



SOURCE: INMOS, The Transputer Databook 2ed. (Redwood Burn LTD, Trowbridge : 1989), pg. 504, Figure 1.1.

Table 12.1. IMS C011 Mode 1 Signal Description.

PIN	*In/Out*	*FUNCTION*
VCC, GND		Power Supply and Return
CapMinus		External capacitor for internal power supply
ClockIn	In	Input Clock (5 MegaHertz)
Reset	In	System reset
SeparateIQ	In	Select Mode and Mode 1 link speed
LinkIn	In	Serial data input channel
LinkOut	Out	Serial data output channel
I0-7	In	Parallel Input bus
IValid	In	Data on I0-7 is valid
IAck	Out	Acknowledge I0-7 data received by link
Q0-7	Out	Parallel Output bus
QValid	Out	Data on Q0-7 is valid
QAck	In	Acknowledge from peripheral that data read

SOURCE: INMOS, The Transputer Databook 2ed. (Redwood Burn LTD, Trowbridge: 1989), pg.505, Tables 2.1 - 2.2.

application notes on the system service signals. Information on the details of the system services such as that the ClockIn signal must be derived from a crystal oscillator instead of an RC oscillator are not covered in this research report.

The mode of operation is selected by the SeparateIQ input signal. Also, both modes of operation support link speeds of either 10 or 20 Megabits per second. Table 12.2 shows the mode selection and speed selection for the IMS C011. The link speed for Mode 2 is determined by another input called Linkspeed. The signal Linkspeed is not part of the Mode 1 signal description.

Table 12.2. Mode Selection for IMS C011.

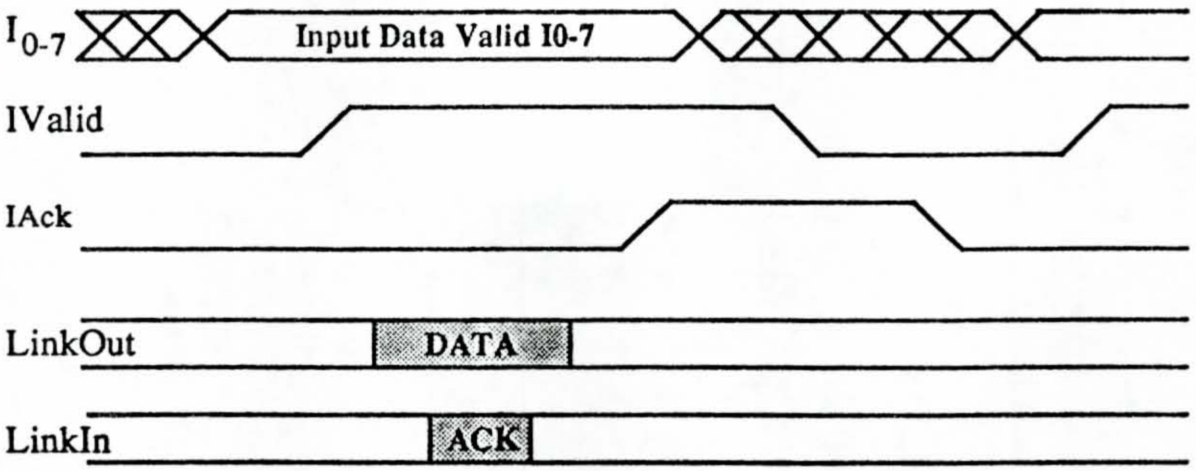
SeparateIQ	MODE	Link Speed Mbits/Sec
VCC	1	10
ClockIn	1	20
GND	2	10 or 20

SOURCE: INMOS, The Transputer Databook 2ed.
(Redwood Burn LTD, Trowbridge: 1989), pg. 508, Table 3.2.

The IMS C011 input port interface is used to transmit bytes of data from a peripheral to a transputer via link. The communication is accomplished by a two-wire handshake provided by the signals IValid and IAck. The peripheral asserts the signal IValid when data is valid on the input port of the IMS C011 to begin communication. When the data has been acknowledged by the appropriate serial link, the IMS C011 asserts the signal IAck to complete the handshake. Recall from Chapter two that each byte has to be acknowledged before the next byte can be transmitted. Refer to Figure 2.2 for the formats of the data and acknowledge packets on the serial links. Figure 12.2 shows a timing diagram for an IMS C011 input port transaction. Refer to the Inmos databook for specific timing parameters.

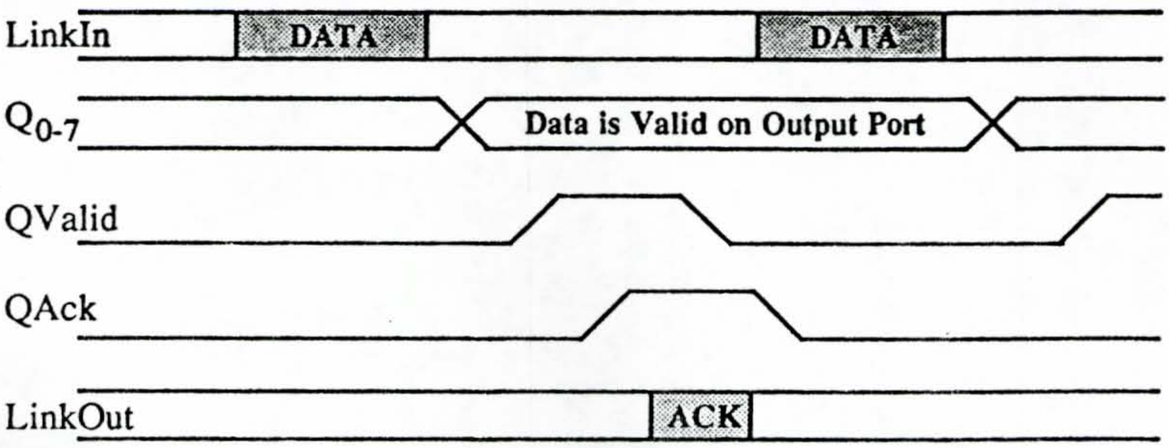
The IMS C011 output port interface is used to convert serial data received on its input link into byte format. When the data is available on the IMS C011 output port, it asserts its QValid signal. The IMS C011 does not accept another serial transmission until an QAck is received from the Output Port Controller. When the QAck signal is received, the IMS C011 shall transmit an acknowledge packet (ACK) on its output serial link. Once an ACK packet is transmitted, the IMS C011 shall deassert its QValid signal and can accept another serial transmission on its input serial link. A timing diagram showing an IMS C011 output port transaction with the Output Port Controller is shown in Figure 12.3.

Figure 12.2. IMS C011 Mode 1 Parallel Data Input to Link Adaptor.



SOURCE: INMOS, The Transputer Data Book 2ed. (Redwood Burn LTD, Trowbridge: 1989), pg. 512, Figure 5.1.

Figure 12.3. IMS C011 Mode 1 Parallel Data Output from Link Adaptor.



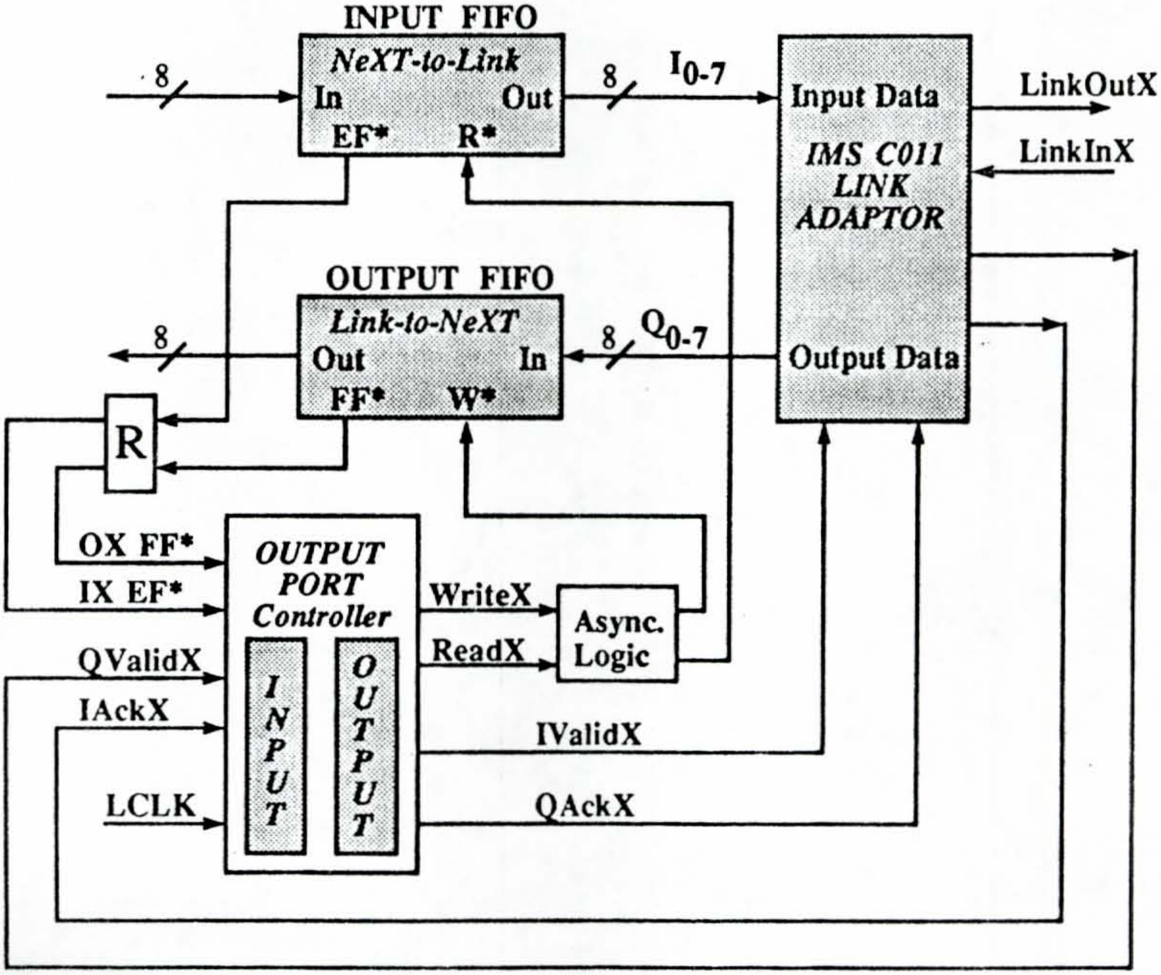
SOURCE: INMOS, The Transputer Data Book 2ed. (Redwood Burn LTD, Trowbridge: 1989), pg. 513, Figure 5.2.

Design Specifications for Output Port Controller

The Output Port Controller shall contain two independent controllers. One controller shall communicate with the input port of the IMS C011 and the other controller with the output port of the IMS C011. A top-level block diagram of the Output Port Controller and its interconnections is shown in Figure 12.4.

The Output Port Controller IMS input sequence shall begin when there is data in the Input FIFO (NeXT-to-Link) to process. The communication shall be asynchronous between the Input FIFO and the IMS C011. The Output Port Controller shall communicate with the IMS C011 via a two-wire handshake. Once the IMS C011 detects a high level on the IValid signal, it shall transmit the byte received on its input

Figure 12.4. Top-level Block Diagram of Output Port Controller Interconnections.



data lines serially on its output serial link. When the byte has been acknowledged by the transputer, the IMS C011 shall asserts its IAck signal. This process continues as long as there is data in the Input FIFO to process. The AHPL description for the Output Port Controller Input IMS Controller is shown in Figure 12.5. The state-diagram for this controller is shown in Figure 12.6. Finally, timing diagrams are presented to show the two different transactions possible. Figure 12.7 shows and IMS C011 input transaction with the Input FIFO Empty Flag (IX_EF*) deasserted throughout the entire transaction. Figure 12.8 shows the same transaction but with the Input FIFO Empty Flag asserted after the first byte has been transmitted to the link adaptor.

The Output Port Controller Output IMS Controller shall write data into the Output FIFO (Link-to-NeXT) when the link adaptor receives a byte from the appropriate transputer. When a byte of data has been received on the input link, the IMS C011 shall assert its QValid signal. The assertion of this signal shall begin the handshaking sequence between the Output Port Controller and the IMS C011. The Output Port Controller IMS Output Controller shall write the byte into the Output FIFO if the FIFO is not Full and assert the QAck signal to conclude the transaction. When the IMS C011 receives the QAck signal, it shall transmit an acknowledge packet on its output link. This sequence is repeated on a Port basis every time a Port input link receives a byte of data. Figure 12.9 shows the AHPL description for the Output Port Controller IMS Output Controller. The state-diagram for this controller is shown in Figure 12.10. Finally, timing diagrams are presented to shown the two different type of transactions which can occur. Figure 12.11 shows an Output IMS sequence with the Output FIFO Full (OX_FF*) flag deasserted throughout the transaction. Figure 12.12 shows an Output IMS sequence with the FIFO Full flag asserted when the QValid signal is asserted by the link adaptor.

This chapter has presented the design of the Output Controller. The Next-to-

Motherboard interface board shall contain four Output Port Controllers (one per port). The purpose of having four ports is so code to be executed in the network of coprocessors can be down loaded faster or so that different nodes of the network can be initialized simultaneously. Each port's Output Port Controller shall begin transmit bytes of data with its appropriate IMS C011 as soon as the Input FIFO Empty Flag is deasserted. Since all port's can be actively transmitting or receiving bytes of data, four Output Controllers and IMS C011's are required on the interface board.

The designer can implement the Output Port Controller any way feasible. The author suggest using a PAL implementation approach The design of the Output Port Controller can fit in one CYPRESS 22V10 PAL (one PAL per port). The ABEL listing for the Output Port Controller implemented in a CYPRESS 22V10 is shown in Appendix C.

Figure 12.5. AHPL Description of Output PORT Controller IMS C011 Input.

```
MODULE:   OUTPUT_CONTROLLER_INPUT_IMS
MEMORY:
INPUTS:   IX_EF*, IACKX, LRESET*.
OUTPUT:   IVALIDX, READX.

"Description of Inputs
"
" IX_EF*    = Registered version of Input FIFO Empty Flag for
"             PORT X, where X = 0 - 3.
" IACKX     = Input Data acknowledge from the IMS C011 Link
"             Adapter of PORT X.
" LRESET*   = Reset pulse from the NBIC.

"Description of Outputs
"
"IVALIDX    = Asserted by controller when data is valid at the
"             input PORT of the IMS C011.
"READX      = Signal used to derive the Read Pulse for the
"             PORT X Input FIFO.

BODY

1. READX = 0;                     " Stay in this state till there is
   IVALIDX = 0;                   " data in the PORT X Input
   --> ( $\overline{IX\_EF*}$ ) / ( 1 ).     " FIFO to process.

2. READX = 1;                     " Assert the READX signal to
   IVALIDX = 0;                   " read a byte from the Input FIFO.

3. READX = 0;                     " Assert IValidX to commence
   IVALIDX = 1;                   " handshaking sequence with IMS C011.
   --> ( $\overline{IACKX}$ ) / ( 3 ).       " Stay in this state till IAckX is asserted
                                  " by the IMS C011 of PORT X.

4. READX = 0;
   IVALIDX = 1;
   --> (IACKX, $\overline{IACKX} \wedge$ IX_EF*, $\overline{IACKX} \wedge \overline{IX\_EF*}$) / (4, 2, 1).
   " Stay in this state till the acknowledge signal from the IMS C011
   " is deasserted.  Once the signal (IAckX) is deasserted, go to
   " State #2 if the FIFO Empty Flag is deasserted to process another
   " byte in the Input FIFO.  If the Empty FIFO Flag is asserted, then
   " go to State #1 and wait for the flag to be deasserted.

END SEQUENCE

   --> ( $\overline{LRESET*}$) / ( 1 ).  "Go to State #1 if Reset pulse is asserted.

END.
```

Figure 12.6.. State-Diagram for Output Controller to IMS C011 Input Sequence.

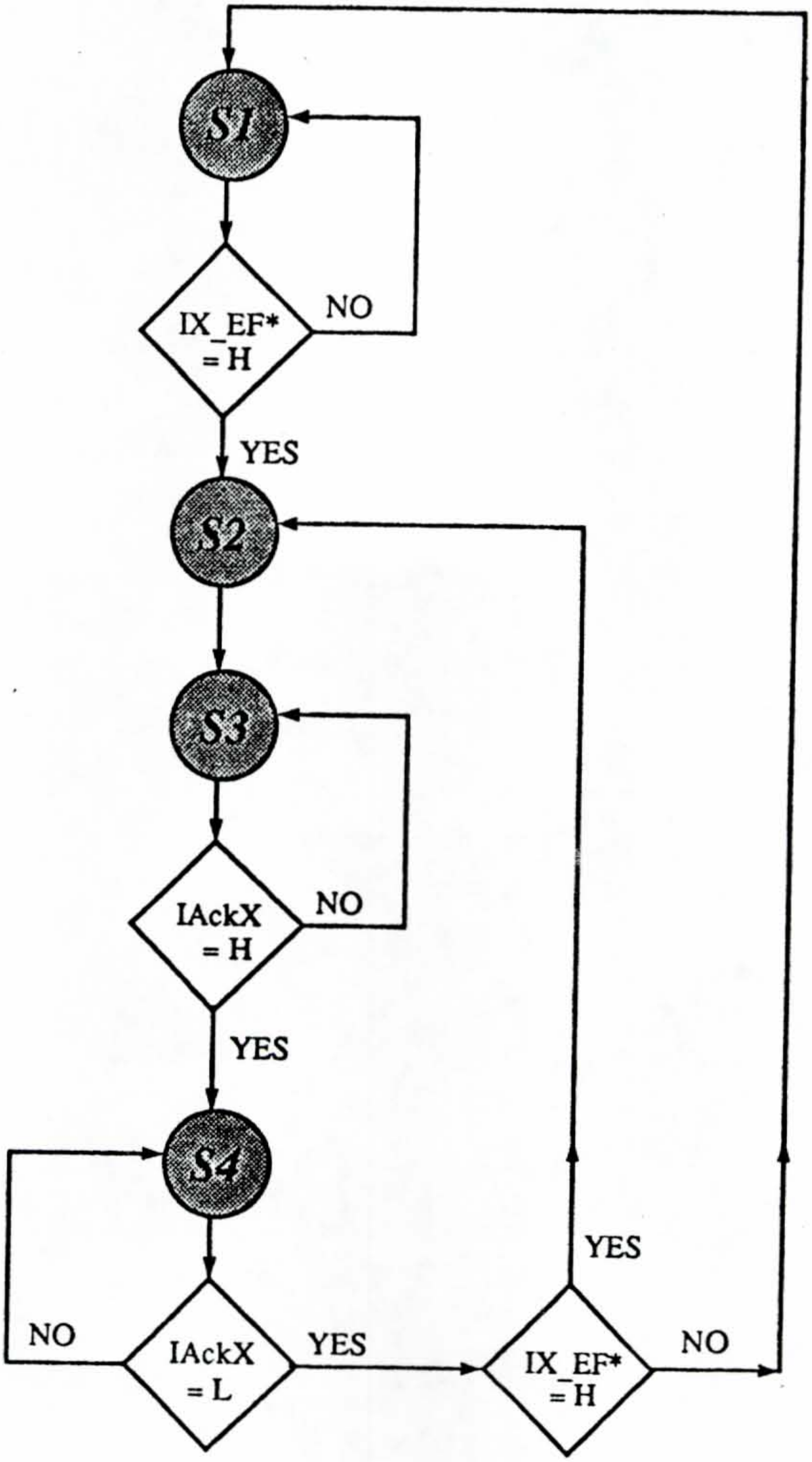


Figure 12.7. Timing Diagram for Output Controller Input Transaction with FIFO Empty Flag Deasserted during Transaction.

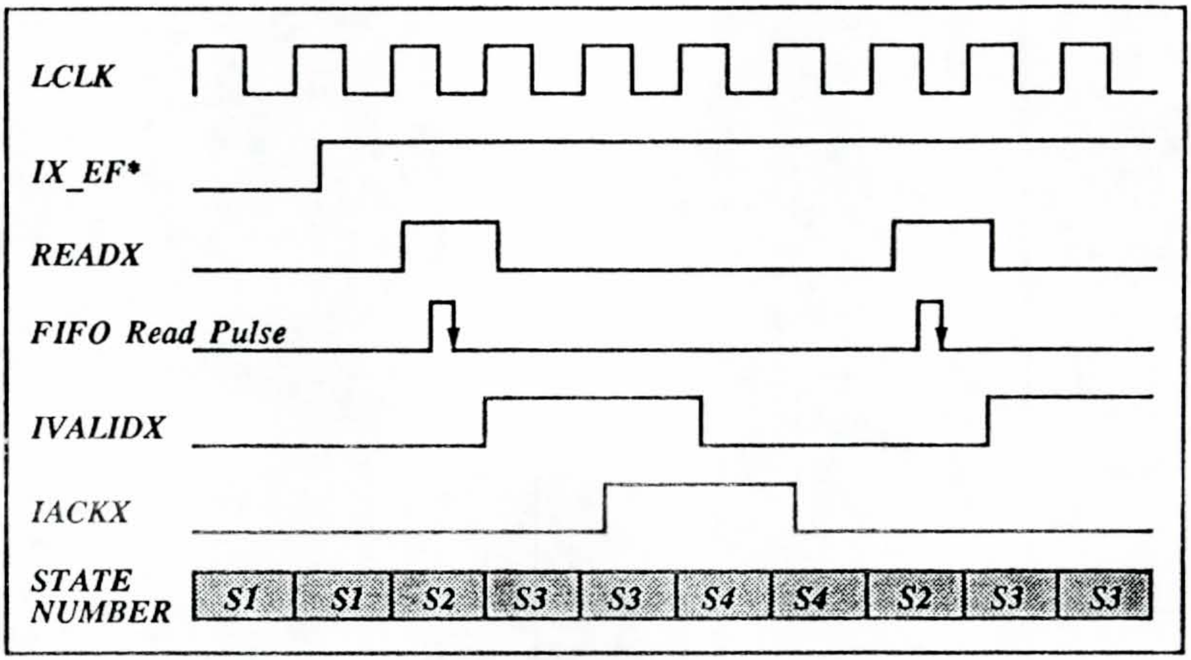


Figure 12.8. Timing Diagram for Output Controller Input Transaction with FIFO Empty Flag Asserted during Transaction.

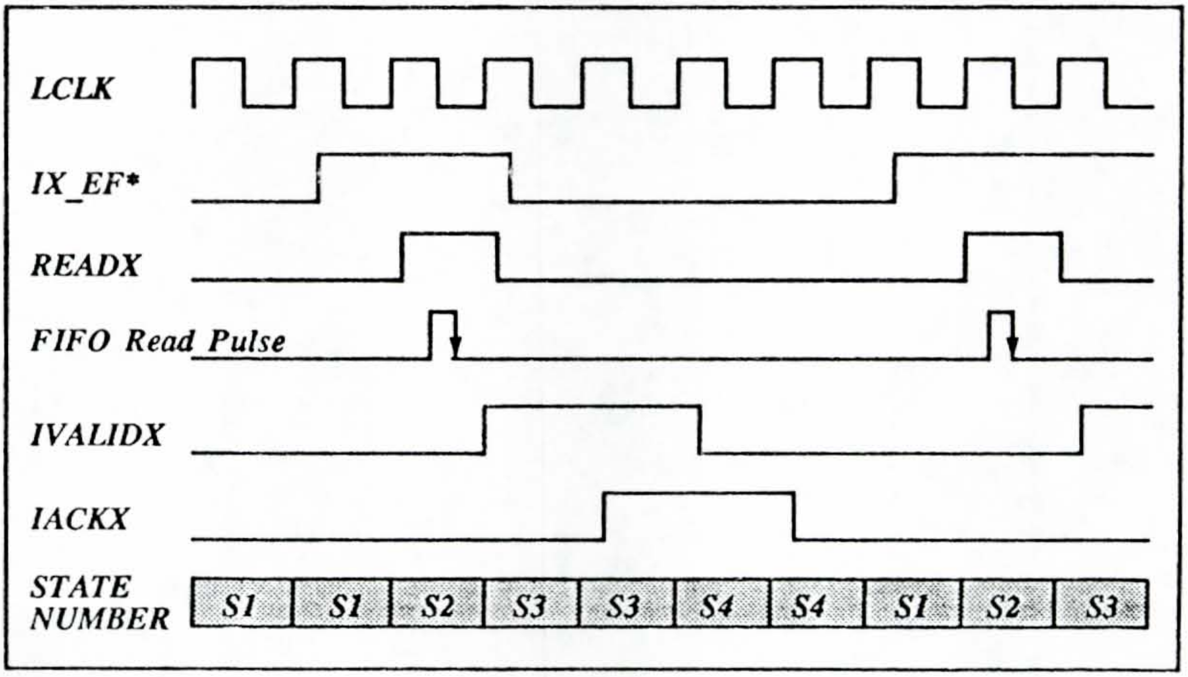


Figure 12.9. AHPL Description of Output PORT Controller IMS C011 Output.

```
MODULE:   OUTPUT_CONTROLLER_OUTPUT_IMS
MEMORY:
INPUTS:   QVALIDX, OX_FF*, LRESET*.
OUTPUT:   QACKX, WRITEX.

"Description of Inputs
"
" QVALIDX  = Asserted by the link adaptor of Port X when there
"             is a valid byte on its output port interface.
" OX_FF*   = Registered version of the Output FIFO Full flag
"            from Port X.
" LRESET*  = Reset pulse from the NBIC.

"Description of Outputs
"
" QACKX    = Asserted by controller when data has been written
"             to the Output FIFO of Port X.
" WRITEX   = Signal used to derive the Write Pulse for the
"             PORT X Output FIFO.

BODY

1.  WRITEX = 0;                      " Stay in this State until the link
    QACKX  = 0;                      " adaptor of Port X asserts the
    --> ( $\overline{QVALIDX}$ ) / ( 1 ).   " QVALIDX signal to begin handshake.


2.  WRITEX = OX_FF*;                 " Stay in this State till the Output FIFO
    QACKX = 0;                       " is ready to accept another byte.
    --> ( $\overline{OX\_FF*}$) / ( 2 ).


3.  WRITEX = 0;                      " Stay in this State till the IMS C011
    QACKX = 1;                       " of Port X deasserts its QVALID signal.
    --> ( QVALIDX, $\overline{QVALIDX}$ ) / ( 3, 1 ).


END SEQUENCE

    --> ( $\overline{LRESET*}$) / ( 1 ).   "Go to State #1 if Reset pulse is asserted.

END.
```

Figure 12.10. State-Diagram for Output Controller to IMS C011 Output Sequence.

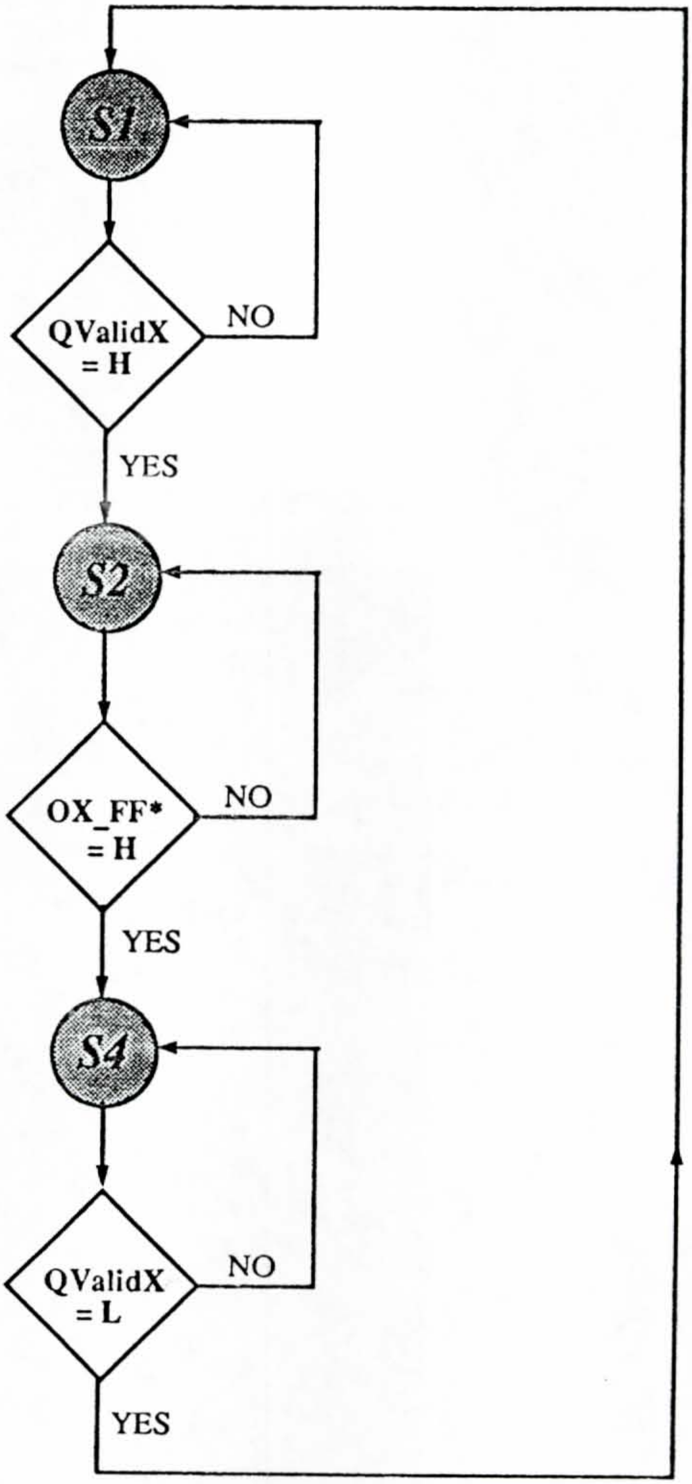


Figure 12.11. Timing Diagram for Output Controller Output Transaction with FIFO Full Flag Deasserted during Transaction.

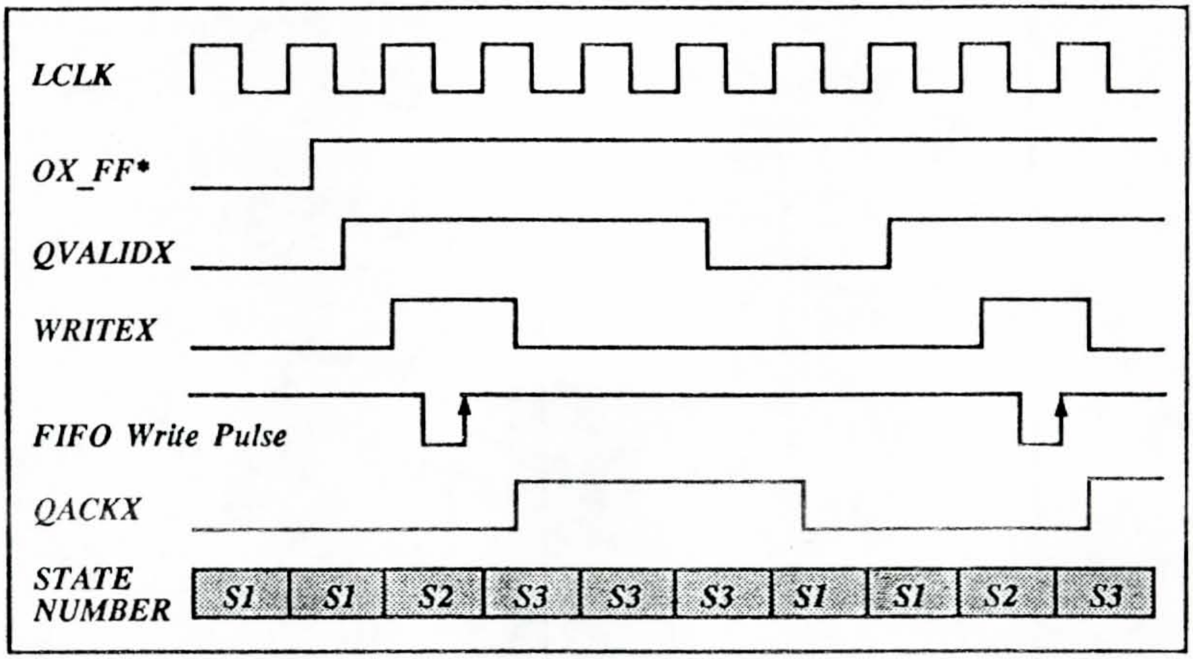
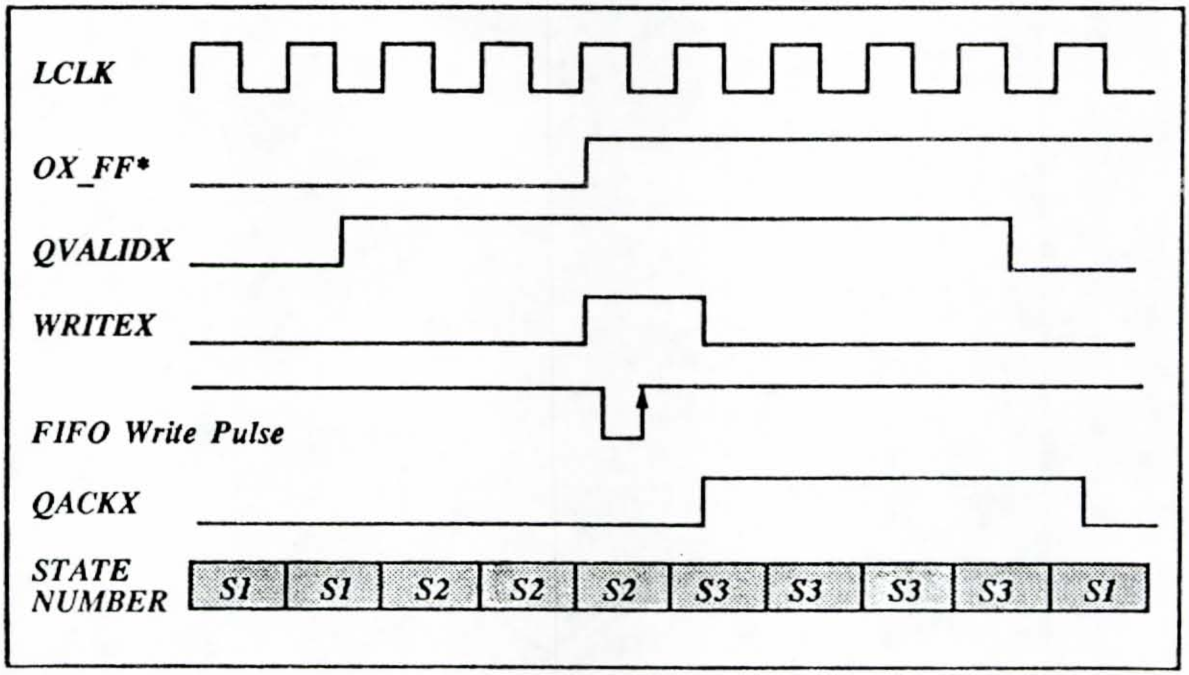


Figure 12.12. Timing Diagram for Output Controller Output Transaction with FIFO Full Flag Asserted during Transaction.



CHAPTER XIII

TOP-LEVEL DESIGN OF ERROR CONTROLLER

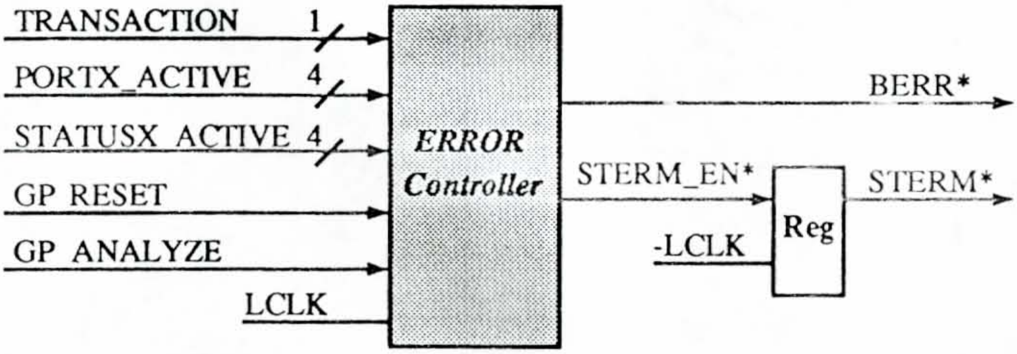
Description of Error Controller

The purpose of this controller is to terminate an invalid transaction requested by the user with an error termination status or acknowledge any Port Reset or Analyze transactions. If the user reads or writes to an undefined address on the board or to a valid address incorrectly, then the Error Controller shall be activated and issue a Bus Error (BERR*) termination signal. For example, suppose the user wants to know if the Input FIFO of PORT #0 is empty. If the user performs a burst read to address 'SX4011XX' (X are don't care bytes and S is the board's slot number), then the address decoder module shall not assert the signal "PORT0_IN_EF_FLG" because a burst read was issued instead of a single-word read. If this case occurs, then the Error Controller needs to be activated to issue a Bus Error termination signal. The BERR* signal shall indicate to the user that the NeXT-to-Motherboard interface board can not place valid data on the bus for the given address. Also, this controller shall terminate (assert signal STERM*) any Port Reset or Analyze transactions issued by the user. This controller can be considered as the watchdog controller of the board. It shall inform the user if an invalid memory address of the board has been referenced.

Top-Level Specifications for Error Controller.

A top-level block diagram of the Error Controller is shown in Figure 13.1. The controller receives all of its inputs from the address decoder module. The TRANSACTION signal is set when a valid Port transaction begins. The PORTX_ACTIVE signals (one per port) shall be set when the user reads or writes to the appropriate Port address. The STATUSX_ACTIVE signals (one per port) shall be set when the user reads to the appropriate special register address of the board's memory address space. If any of the four Status signals are set, then the user is requesting FIFO status or Subsystem Error status from the board. If any of the Port active or Status active signals are set, then the Error Controller shall not issue a Bus Error signal.

Figure 13.1. Top-Level Block Diagram of Error Controller.

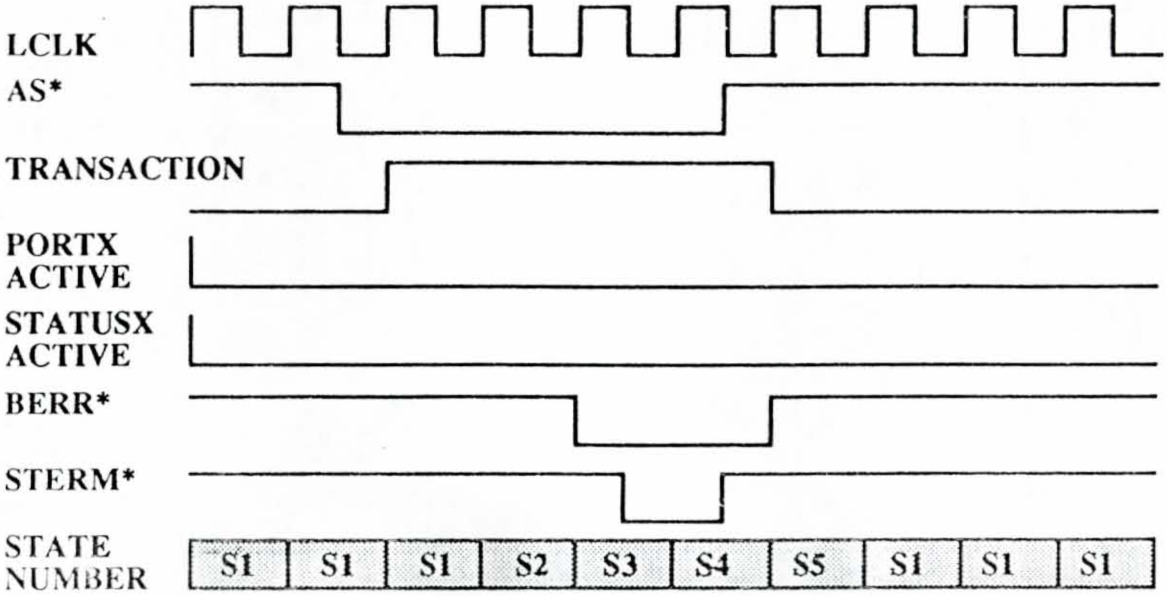


The AHPL description for the Error Controller is shown in Figure 13.2. This is the only AHPL description which asserts the BERR* signal. However, the AHPL description for the Input Port Controller also asserts the synchronous termination signal STERM*. The designer needs to combine the Error Controller's STERM_EN* with the Input Controller's STERM_EN* signal to generate a common STERM_EN* signal. This can be accomplished by a logical **"AND"** of the two STERM_EN* signals. If a controller (Input or Error) is not activated, then it shall deassert its STERM_EN* signal. If any of the STERM_EN* signals are asserted, then the common STERM_EN* signal shall also be asserted. Two timing diagrams for the Error Controller AHPL description are shown in Figures 13.3 and 13.4. The timing diagram of Figure 13.3 shows an invalid Port Transaction since none of the active signals from the address decoder are asserted. The transaction shown in Figure 13.3 results in a Bus Error being issued by the Error Controller. The second timing diagram shows the sequence of states executed for a valid Port Transaction.

Figure 13.2. AHPL Description for Error Controller.

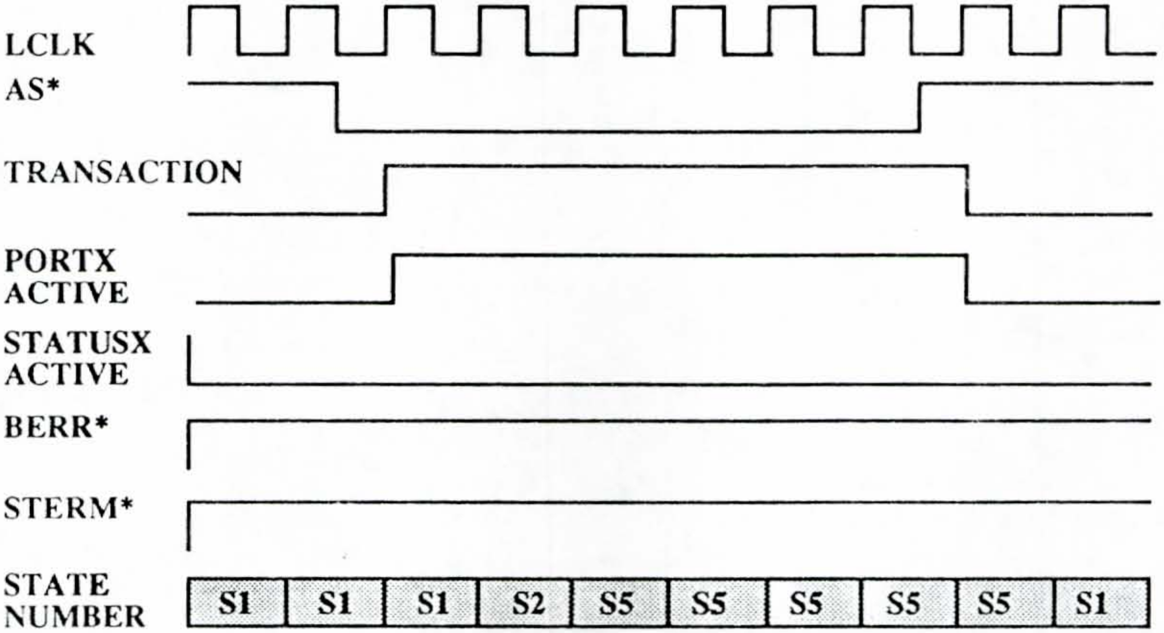
```
MODULE:  ERROR_CONTROLLER.
MEMORY:  ERROR.
INPUTS:  TRANSACTION, LRESET*, PORT0_ACTIVE, PORT1_ACTIVE,
         PORT2_ACTIVE, PORT3_ACTIVE, STATUS0_ACTIVE,
         STATUS1_ACTIVE,  STATUS2_ACTIVE, STATUS3_ACTIVE,
         GP_RESET,  GP_ANALYZE.
OUTPUTS: BERR*, STERM_EN*.

"Description of Inputs
"
"TRANSACTION       = Set by Address Decoder when a valid Port transaction
"                     is to take place.  (Not set for a Port Reset or Analyze).
"LRESET*           = Asserted by NBIC when a reset is to take place.
"PORTX_ACTIVE      = Set by Address Decoder when a Read or Write
"                    transaction to Port X is to take place (X = 0 - 3).
"STATUSX_ACTIVE    = Set by Address Decoder when a FIFO Status Flag or
"                    or Subsystem Error flag has been requested for Port X.
" GP RESET         = Asserted by the Address Decoder when any of the
"                     four ports is to be reset.
" GP ANALYZE       = Asserted by the Address Decoder when any of the
"                     four ports is to be analyzed.
"Description of Outputs
"
"BERR*             = Set by Controller to issue a synchronous 32-bit bus error.
"STERM_EN*         = Asserted by controller to terminate transaction.

BODY

1.   --> ( $\overline{\text{TRANSACTION}}$ ) / (1).      "Stay in this State until a Port transaction
     BERR* = 1; STERM_EN* = 1.         " is to take place.  Deassert all outputs

2    --> ( ACTIVE ) / ( 5 ).           "Go to State #5 if the ACTIVE signal
     BERR* = 1; STERM_EN* = 1.         "is asserted.  Deassert all outputs

3.   BERR* = RESET  ;                  "Assert Bus Error if not reset
     STERM_EN* = 0.                    "assert synchronous termination signals.

4.   BERR* =  RESET ;                  "Assert Bus Error if not reset and
     STERM_EN* = 1.                    "deassert synchronous termination signal.

5.   --> ( TRANSACTION, $\overline{\text{TRANSACTION}}$ ) / ( 5, 1 ).   " Stay in this state till
                                                       " Transaction signal is
END SEQUENCE                                           "  deasserted."

   ACTIVE = ( PORT0_ACTIVE ∨ PORT1_ACTIVE ∨ PORT2_ACTIVE ∨
              PORT3_ACTIVE ∨ STATUS0_ACTIVE ∨ STATUS1_ACTIVE ∨
              STATUS2_ACTIVE ∨ STATUS3_ACTIVE );

   RESET = GP_RESET ∨ GP_ANALYZE ;

   --> ( $\overline{\text{LRESET*}}$ ) / ( 1 ).   " Go to State #1 if local Reset pulse is
                              "asserted by NBIC.
END.
```

Figure 13.3. Timing Diagram for an Invalid Port Transaction..



NOTES: Assumes GP_RESET and GP_ANALYZE signals are deasserted.

Figure 13.4. Timing Diagram for a Valid Port Transaction.



NOTES: Assumes GP_RESET and GP_ANALYZE signals are deasserted.

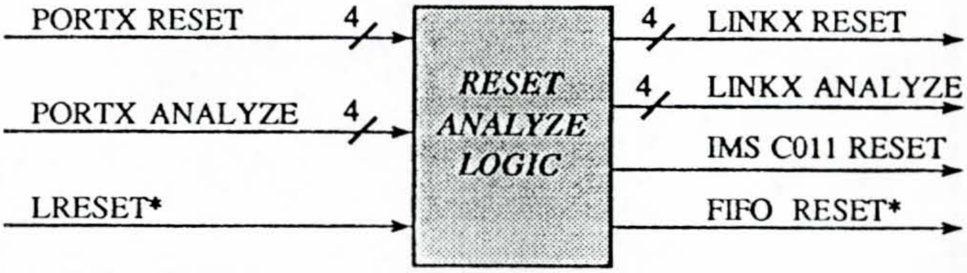
CHAPTER XIV

RESET, ANALYZE AND INTERRUPT SIGNALS

Description of Reset and Analyze Signals

The purpose of this chapter is to describe all the reset, analyze and interrupt signals which are required on the NeXT-to-Motherboard interface board. All the reset and analyze signals shall be represented in terms of a timing diagram. No AHPL description or state-diagrams shall be presented in this chapter. A top-level block diagram of the inputs to the reset and analyze logic is shown in Figure 14.1. The signals PortX_Reset and PortX_Analyze are asserted by the address decoder module. The PortX_Reset signal (one per port) is asserted when the user selects to reset the root transputer of a Port X. Similarly, the PortX_Analyze signal is asserted when the user selects to analyze the state of the transputer network of Port X. The assertion of the signals is accomplished by writing to the appropriate special register address on the interface board (listed in Table 8.1). The FIFO and IMS C011 reset signals are asserted by the reset logic every time the LRESET* is asserted by the NBIC. The FIFO reset signal is used to reset all the Input and Output FIFO's on the interface board. Similarly, the IMS C011 reset signal is used to reset all the link adaptors on

Figure 14.1. Top-level Block Diagram of Reset and Analyze Logic.



the board. The next section shall introduce timing diagrams for a Link Reset, Link Analyze, FIFO or Link Adaptor (IMS C011) reset sequence (Inmos, "The Transputer Databook).

The trailing edge of the reset pulse to the transputer (LINKX RESET) shall initialize the transputer, trigger the internal and external memory configuration and begin the bootstrap sequence. All of the events described above are performed in sequence instead of in parallel. The Analyze signal is used in conjunction with the Reset signal to halt the root transputer and allow the user to examine the internal states of the transputer so the cause of an error may be determined. The state of the Analyze signal indicates if the transputer is to be analyzed or reset. If the Analyze signal is asserted, then the transputer flags are not altered and the processor shall halt at the next descheduling point. The Reset signal must be asserted and deasserted some time after the Analyze signal has been asserted. By asserting and deasserting the Reset signal, the root transputer shall not performed the memory configuration sequence nor the refresh cycles which follow; therefore, the previous memory configuration shall be used for any external memory accesses. If the Analyze signal is deasserted before the Reset signal is asserted, then the state and operation of the transputer are undefined. If the signal BootFromRom is high, then the transputer shall executed its boot program in ROM at the trailing edge of the Analyze signal. Otherwise, the transputer shall wait for a control byte after the trailing edge of the Analyze signal. Figure 14.2 shows a transputer Reset timing diagram and Figure 14.3 shows a transputer Analyze timing diagram. From the timing diagrams, it can be seen that the minimum time that the reset pulse is maintained at a high level depends on the transputer clock period denoted by the variable ClockIn. Recall, that the Error Controller shall issue a synchronous termination signal (STERM*) when the user performs a link reset or analyze sequence.

Figure 14.2. Transputer Reset Timing Diagram.

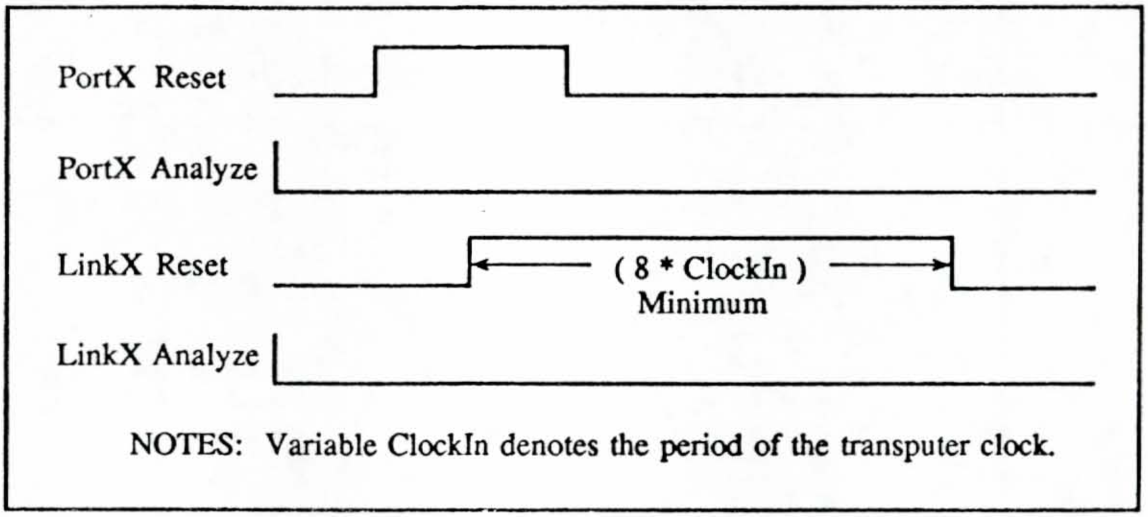
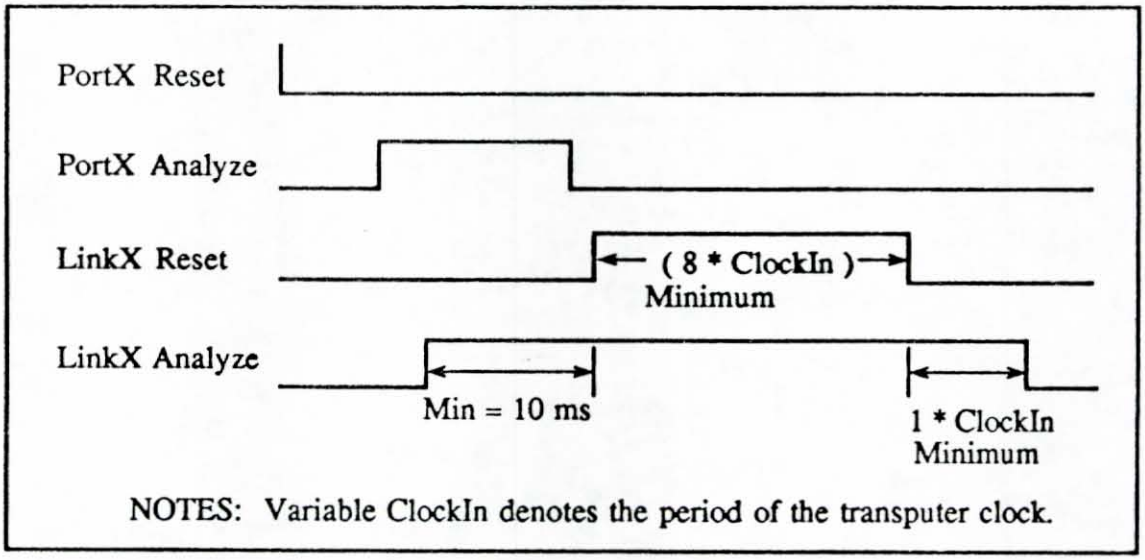
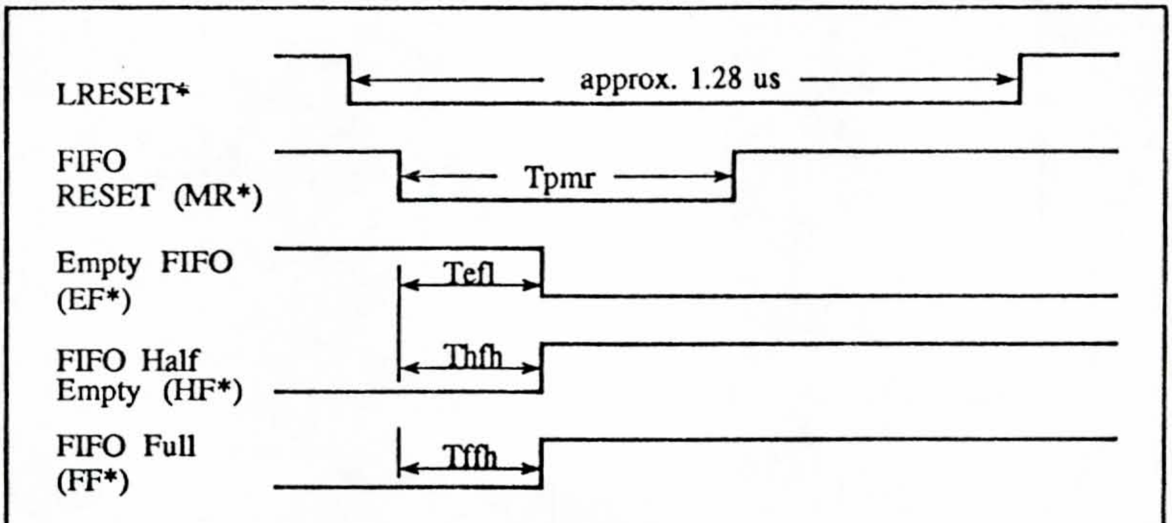


Figure 14.3. Transputer Analyze Timing Diagram.



The next reset signal required is for all the Input and Output FIFO's on the board. This signal can probably be a buffered version of the LRESET* signal. Figure 14.4 shows a FIFO reset timing diagram. The state of the FIFO flags are shown in this timing diagram for completeness. The timing parameters shown in Figure 14.4 are defined in Table 14.1. The last reset signal to be defined is the IMS C011 reset.

Figure 14.4. FIFO Reset Timing Diagram.



Source: CYPRESS, Semiconductor BiCMOS/CMOS Data Book, published March 1, 1990, page 5-33, No figure number.

Table 14.1. Timing Parameters for FIFO Reset Transaction.

Para-meter	7C4XX-20		7C4XX-25		7C4XX-30		7C4XX-40		7C4XX-65	
	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX
Tpmr	20		25		30		40		65	
Tefl		30		35		40		50		80
Thfh		30		35		40		50		80
Tffh		30		35		40		50		80

NOTES: All times shown above are in nanoseconds. The information for this table was obtained from the CYPRESS Semiconductor BiCMOS/CMOS Data Book, published March 1, 1990.

The IMS C011 needs to be reset every time the LRESET* signal is asserted by the NBIC. After power is applied to the board, the clock input of the IMS C011 needs to be operating at least 10 milliseconds (ms) before the trailing edge of the reset pulse. The minimum pulse width of the IMS C011 reset pulse is also dependent on the frequency of its clock. The minimum assertion time for the IMS C011 reset pulse is eight times the period of the clock (8 * ClockIn). When the IMS C011 is reset, the signal Linkout is held at a low state, the handshaking lines IAck and QValid are held low and the output port data lines (Q0-7) are undefined. The last section of this chapter shall explain the generation of the local bus interrupt signal SINT*.

Description of the Interrupt Signal

The Inmos Host Interface guidelines specifies that the interface board shall have the capability to interrupt the host processor. The NeXT-to-Motherboard interface board shall assert the local interrupt signal SINT* when any of its four Output FIFO's (Link-to-NeXT) have data to be processed. When bytes of data are received on any of the four link inputs (LinkInX), the appropriate Output Port Controller shall write the byte into the appropriate Output FIFO. After the data is written, the empty FIFO flag for that FIFO shall be deasserted. The four empty FIFO flags should be "NORed" to form the active low interrupt signal SINT*. When the SINT* signal is asserted and the mask interrupt register is enabled, the NBIC shall generate a NeXTbus interrupt signal. The host software can enable or disable the mask interrupt register depending on the type of operation being performed. If the host is expecting return data from the array of coprocessors, then the host software shall enable the appropriate bit in the mask interrupt register so the interface board can alert the host when data is valid in the Output FIFO. Once the interrupt is acknowledged by the host software, the bit in the mask interrupt register should be cleared. The reason for clearing the bit is that the NBIC shall continue to assert the NeXTbus interrupt signal since the empty FIFO flag should still be deasserted until all the data has been read from the Output FIFO. The assertion of the interrupt signal SINT* shall indicate to the host that there is data from the network of coprocessor to be processed. It does not inform the host software which of the ports have the data. The host software should be smart enough to know which of the ports should be expecting return data.

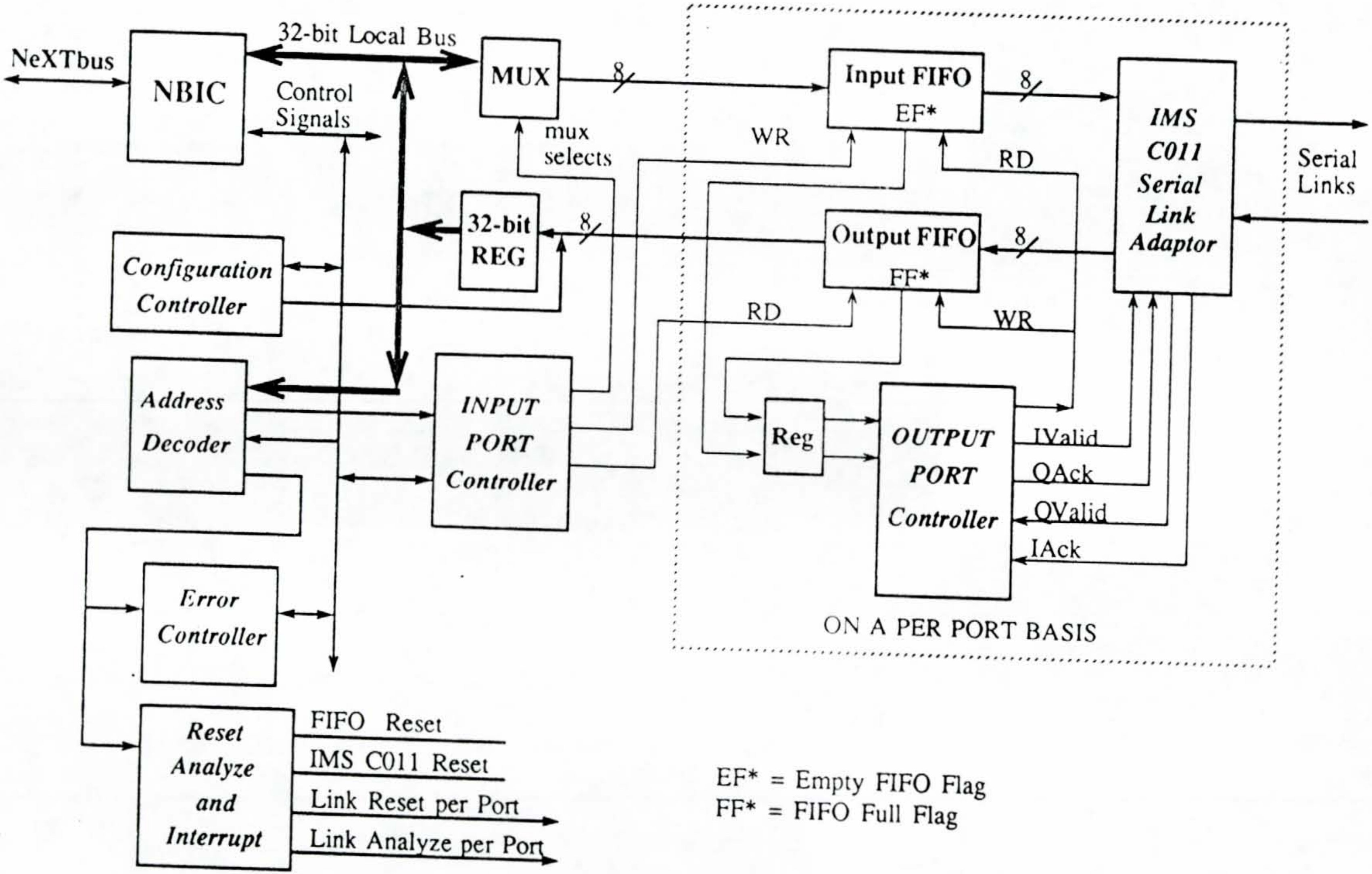
CHAPTER XV
CONCLUSIONS

This research report has presented a top-level design using the NBIC for a NeXT-to-Motherboard interface board. The design of the interface board is composed of the following six hardware modules: Configuration Controller, Address Decoder, Input Port Controller, Output Port Controller, Error Controller and Reset, Analyze and Interrupt logic. The Configuration Controller shall initialize all the appropriate registers in the NBIC. The Address Decoder module shall latch the address at the beginning of a transaction and assert the appropriate control signals for the Port Controllers, Error Controller, Reset and Analyze logic. The Input Port Controller is responsible for all transactions between the NBIC local bus and the input/output FIFO's. The Output Port Controller is responsible for all transactions between the input/output FIFO's and the Inmos serial link adaptors (IMS C011). The Error Controller shall terminate a transaction with an error status when an invalid memory address on the board has been referenced by the user or terminate a Port Reset or Analyze transaction with a 32-bit acknowledge status. Finally, the reset logic generates all the required reset signals for the interface board and asserts the appropriate link reset signal (on a port basis) when requested by the user. The link reset signal is used to reset the root transputer in the motherboard connected to that particular port. The Analyze logic asserts the appropriate analyze and reset signal (on a port basis) when requested by the user. The analyze and reset signals in a transputer are used to halt the processor so the cause of a system error can be identified. The internal states of a transputer can be analyzed when the processor has been halted by an analyze sequence. The

Interrupt logic asserts the local interrupt signal (SINT*) which is used by the NBIC to generate a global interrupt signal on the NeXTbus.

All of the modules described in the previous paragraph have been designed in AHPL. Also, timing diagrams have been generated for all of the modules and state diagrams have been generated for several of the modules. The top-level design has been performed on a module basis so that modularity can be achieved on a per port basis. A top-level block diagram of all the hardware modules and their interconnections is shown in Figure 15.1. The board has been initially designed with four ports which shall allow four motherboards or nodes of a parallel network to be initialized. If more than four ports are required, then extra port hardware needs to be added and the Address Decoder, Input Port Controller, Error Controller and Reset modules need to be modified to support the extra port(s). Each port hardware consists of an Input FIFO, Output FIFO, Output Port Controller and an Inmos serial link adaptor as shown in Figure 15.1. If less than four ports are required, then only the required number of port hardware is required and the appropriate control input signals of the Input Port Controller, Error Controller and Reset modules needs to be deasserted. The Input Port Controller, Address Decoder, Error Controller and Reset, Analyze and Interrupt modules only need to be modified if more than four ports are required. The author selected four ports has a starting number for the design. The algorithm presented in this research report can be modified to support any number of ports. The actual number of ports on the board shall depend on the specific application of the board and the amount of real estate remaining on the NeXTbus card after the detail design of the Configuration Controller, Address Decoder, Error Controller, Input Port Controller and the Reset, Analyze and Interrupt Logic is complete. The real estate remaining on the board shall allow the designer to determine the maximum number of port hardware (Input FIFO, Output FIFO and Output Port Controller) which the board can support. If the maximum number of ports is greater than four, then extra real estate might be

Figure 15.1 All Hardware Modules and their Interconnections.



EF* = Empty FIFO Flag
FF* = FIFO Full Flag

required because several of the modules required modifications to support the extra port(s).

The design of the NeXT-to-Motherboard interface board is based on using the NeXTbus Interface Chip (NBIC) from NeXT Corporation. There are advantages and disadvantages in using the NBIC in the design of the interface board. An obvious advantage in using the NBIC is that it reduces the amount of design time and logic required to interface to the NeXTbus since it provides a single-chip interface to the NeXTbus. A disadvantage of the NBIC is that it only supports a burst transfer size of four words during burst transactions. The NBIC does not fully support the NeXTbus protocol since the NeXTbus supports a maximum burst transfer size of 32 words. The reason that the NBIC only supports a transfer size of four words during burst transactions is due to the limitation of silicon real estate. Recall, the NBIC has two internal FIFO's where each FIFO can buffer two transactions. Each transaction stored in the FIFO consists of one address and either one or four data words. The NBIC contains two internal FIFO's since it supports both Master and Slave operations. If NeXT Corporation had designed a NBIC for Slave only applications, then the NeXTbus Master/Local Slave Transaction FIFO shown in Figure 5.5 (page 45) can be deleted from the chip. The deletion of this FIFO shall create extra silicon real estate which shall expand the Local Master/NeXTbus Slave Transaction FIFO to buffer a minimum of eight data words during burst transactions (maybe more depending how much control logic can be deleted from the chip). Another disadvantage about the NBIC is that it does not generate an error when a burst transfer size of greater than four is attempted over the NeXTbus. After talking to a technical representative from NeXT, it is believed that only the first four words of the transaction are stored in the appropriate transaction FIFO. The remaining words are acknowledged but never stored in the FIFO. **When performing burst transactions with the NBIC, the burst transfer size must always be four words.** The only solution to correct the limitations of the NBIC is to design custom logic which shall support the NeXTbus protocol completely.

The design of custom interface logic shall increase design time and logic required. An advantage of designing a custom interface is that the user shall have the capability of selecting different burst transfer sizes (up to a maximum of 32 words). Also, processing speed of the board shall improve for any burst transfer size greater than four words. A disadvantage of designing a custom interface is that the number of ports on the board shall be reduced due to the extra real estate consumed by the extra logic. The decision between using the NBIC or designing a custom interface depends on several factors which need to be defined. First, is the speed of the interface acceptable using the NBIC with a maximum burst transfer size of four words ? The answer to this questions depends on the number of words to be transferred across the interface and the period of the local bus clock (LCLK) selected by the designer of the board. Code can be downloaded quicker with a burst transfer size of 32 words (one transaction equals 32 words) versus a burst transfer size of 4 words (eight transactions shall equal 32 words). Recall, every transaction has an address phase, data phase and acknowledgment phase. Extra clocks are wasted due to the address and acknowledgement phase when a small burst transfer size is used. Second, what is the maximum number of ports which shall be required for the interface board ? After these two questions are answered, the designer can determine which approach (NBIC versus Custom Logic Interface) shall support the two criterias. The author has assumed that the performance achieved by using the NBIC is acceptable in the design of the interface board. If custom logic is to be designed, an attempt should be made to have a local bus interface similar to the NBIC so the amount of redesign time for several of the hardware modules defined in this research report can be reduced.

CHAPTER XVI

SUMMARY OF WORK REMAINING

The purpose of this chapter is to state all the task which the author believes remain to be performed. This research report has define the ground work for the detail design of the NeXT-to-Motherboard interface board. It has describe the modules of the board in AHPL so the designer has total freedom of implementation. The only issue which was not addressed by this report is testability of the board. This is the first task which the author believes needs to be addressed.

The task of the board's testability requires the current AHPL descriptions to be modified so data can be written and read from the NeXTbus without the coprocessor architecture connected to the interface board. This task can be accomplished by different methods. One method is by using one of the undefined address bits (LAD[27:24], LAD[7]) to indicate a test transaction. Recall, that an address between sX0000XX through sX0FFFXX (s is the slot number of the board) shall be processed by Port #0 and that address bits 2E00 through 2E06 have been previously defined by NeXT. When a write transaction (single-word or burst) is performed with the new address bit enabled, the Input Port Controller shall perform the write transaction specified and enable a test mode bit on the board. However, the Output Port Controller shall read data from the Input FIFO and write it to the Output FIFO instead of to link when the test mode bit is asserted. After an interrupt is generated by the interface board, the user can then perform a read transaction (single-word or burst) with the new test address bit asserted to read the data previously written. The read is performed with the new address bit asserted so that the test mode bit on the board remains asserted This new test mode bit can be added to state one of the address decoder AHPL description. A multiplexor shall be used to select between the Input FIFO data lines or the IMS C011 link adaptor output port data lines. The multiplexor shall select the Input FIFO data lines when the test mode bit is enabled. Otherwise, the multiplexor shall select the output data lines from the link adaptor. By adding this new address bit, all the FIFO's on the board can be tested. Blocks of data written to the Input FIFO's would eventually be written into the Output FIFO's. If the blocks of data written into the Input FIFO of Port X compares to the blocks of data read from the Output FIFO of Port X, then read/write access to Port X has been tested.

Along with the new AHPL descriptions to be redesigned, an interface software program needs to be developed to test the board. The software program needs to have the following capabilities: Write (single-word or burst) known data patterns of arbitrary size into any of the four ports; Read (single-word or burst) data of arbitrary size from any of the ports; Obtain FIFO status (Empty FIFO, FIFO Full, FIFO Half Empty) information from any of the FIFO's on the board; Perform a write/read transaction of arbitrary size on any of the four ports. A write/read transaction shall consists of writing a user supplied number of bytes into the Input FIFO of Port X and comparing the bytes to the bytes read from the Output FIFO of Port X. The program should report to the user the number of bytes which were compared and the total number of failures. If byte failure(s) exists, then the program should display a list of the bytes written to the Input FIFO versus the bytes read from the Output FIFO. Data patterns should be selected so that every bit in a byte is toggled at least once.

After the testability of the board has been designed, the detail design and simulation of the interface board can begin. The detail design/simulation phase shall prove the validity of the AHPL descriptions or improve upon the algorithms described in this research report. If the board is to be designed on a CAD system, logic and simulation models for the NBIC will need to be modeled. If the simulation model for the NBIC can not be developed, then simulation can be performed assuming the inputs to the board are the local bus signals of the NBIC. By assuming the local bus signals of the NBIC as the inputs to the interface board, all the AHPL descriptions described in this research report can be tested. During the detail design phase, the period of the local bus clock (LCLK) needs to be determined. Recall, that the board needs a crystal 5 MegaHertz (MHz) oscillator for the four IMS C011 link adaptors. Also, the connector and cables between the interface and the Motherboard of the array of coprocessor needs to be defined. The connector(s) on the interface board should follow the standard TRAM pinouts.

APPENDIX A. NeXTbus Backplane Pinout.

PIN #	PIN NAME
A1	+12V
A2	+12V
A3	+12V
A4	+12V
A5	SP0*
A6	SP1*
A7	RESERVED
A8	ACK*
A9	TM0*
A10#	INT*
A11	MCLKSEL*
A12#	ARB0*
A13#	ARB1*
A14#	ARB2*
A15#	ARB3*
A16#	RQST*
A17#	RESERVED
A18	TM1*
A19	DRQ*
A20	SPV*
A21	RESERVED
A22	DSTB*
A23	RESET*
A24	RESERVED
A25	START*
A26	RESERVED
A27	BUSCLK
A28	SP2*
A29	SP3*
A30	RESERVED
A31	PON
A32	PUP

PIN #	PIN NAME
B1	ADO*
B2	AD1*
B3	AD2*
B4	AD3*
B5	AD4*
B6	AD5*
B7	AD6*
B8	AD7*
B9	AD8*
B10	AD9*
B11	AD10*
B12	AD11*
B13	AD12*
B14	AD13*
B15	AD14*
B16	AD15*
B17	AD16*
B18	AD17*
B19	AD18*
B20	AD19*
B21	AD20*
B22	AD21*
B23	AD22*
AB4	AD23*
B25	AD24*
B26	AD25*
B27	AD26*
B28	AD27*
B29	AD28*
B30	AD29*
B31	AD30*
B32	AD31*

PIN #	PIN NAME
C1	-12V
C2	-12V
C3	-12V
C4	-12V
C5	GND
C6	GND
C7	VCC
C8	GND
C9	VCC
C10	GND
C11	VCC
C12	GND
C13	VCC
C14	GND
C15	VCC
C16	GND
C17	VCC
C18	GND
C19	VCC
C20	GND
C21	VCC
C22	GND
C23	VCC
C24	GND
C25	GND
C26	VCC
C27	GND
C28	GND
C29	SID28
C30	SID29
C31	SID30
C32	SID31

# 220 Ohm pullup resistor

APPENDIX B. NBIC Pinout Definition.

A1	GTM1*	C7	GND	H13	VCC	N10	SIZ0
A2	GAD18*	C8	GSTART*	H14	LAD17	N11	GND
A3	SPARE	C9	VCC	H15	LAD16	N12	STERM*
A4	GAD20*	C10	GND	J1	GMCLKSEL*	N13	NCS*
A5	GDSTB*	C11	GAD29*	J2	GAD9*	N14	LAD30
A6	SPARE	C12	LAD0	J3	GND	N15	LAD26
A7	GAD23*	C13	LAD3	J13	GND	P1	GAD5*
A8	GAD24*	C14	LAD5	J14	LAD20	P2	GAD4*
A9	GAD25*	C15	LAD8	J15	LAD18	P3	GAD2*
A10	GBUSCLK0	D1	GARB3*	K1	GMCLKSELO*	P4	GAD0*
A11	GAD26*	D2	GAD15*	K2	GINT*	P5	GMASTER*
A12	GAD27*	D3	GND	K3	VCC	P6	LBG/EXSEL
A13	GAD28*	D13	GND	K13	LAD23	P7	GSLAVE*/SINT*
A14	GAD30*	D14	LAD7	K14	LAD22	P8	BR*
A15	LAD1	D15	LAD9	K15	LAD19	P9	RMC*
B1	GRESET*	E1	GARB2*	L1	GAD8*	P10	R/W*
B2	GAD17*	E2	GAD14*	L2	GTM0*	P11	AS*
B3	GSPV*	E3	GND	L3	VCC	P12	DSACK1*
B4	GAD19*	E13	LAD6	L13	LAD27	P13	BACK*
B5	GDSTBO*	E14	LAD10	L14	LAD25	P14	OE
B6	SPARE	E15	LAD11	L15	LAD21	P15	LAD29
B7	GAD22*	F1	GARB1*	M1	GAD7*	Q1	GSP0*
B8	SPARE	F2	GAD13*	M2	GAD6*	Q2	GAD1*
B9	GBUSCLK	F3	VCC	M3	GND	Q3	LCLK
B10	GSP2*	F13	GND	M13	LAD31	Q4	GBCYC*
B11	GSP3*	F14	LAD12	M14	LAD28	Q5	LBR*
B12	BUSCLKIN	F15	LAD14	M15	LAD24	Q6	HALT*
B13	GAD31*	G1	GAD11*	N1	GACK*	Q7	BERR*
B14	LAD2	G2	GAD12*	N2	SPARE	Q8	BG*
B15	LAD4	G3	GND	N3	GSP1*	Q9	DS*
C1	GRQST*	G13	VCC	N4	GAD3*	Q10	FB
C2	GAD16*	G14	LAD13	N5	CPUCLK	Q11	SIZ1
C3	GDRQ*	G15	LAD15	N6	GND	Q12	ASIN*
C4	VCC	H1	GAD10*	N7	BGACK*	Q13	DSACK0*
C5	GND	H2	GARB0*	N8	BREQ*	Q14	PON
C6	GAD21*	H3	GND	N9	VCC	Q15	LRESET*

APPENDIX C. ABEL PAL Implementation of Ouput Port Controller using a 22V10.

```
module   output_controller   flag '-r3', '-t2'

title
'Output Port Controller for
 NeXT-to-Motherboard Interface board
 A. Alvarez        September 26, 1990'


Output_controller     device     'P22v10'

"-----------------------------------------------------------------
" DESCRIPTION:
"
" This PAL is used to transfer bytes of data between the
" IMS C011 Link Adaptor and the Output FIFO or between
" Input FIFO and the IMS C011.  All transfers are performed
" via a two-wire hand" shake.
"
"-----------------------------------------------------------------

" Constants:

      ON   = 1;
      OFF  = 0;
      H    = 1;
      L    = 0;
      X    = .X.;
      C    = .C.;

" State Definitions for Output Controller.

      ST_OA  = ^b000 ;  " State #1 of AHPL Description of Figure 12.9
      ST_OB  = ^b001 ;  " State #2 of AHPL Description of Figure 12.9
      ST_OC  = ^b010 ;  " State #3 of AHPL Description of Figure 12.9
      ST_OC  = ^b011 ;  " Unused State

      ST_IA  = ^b000 ;  " State #1 of AHPL Description of Figure 12.5
      ST_IB  = ^b001 ;  " State #2 of AHPL Description of Figure 12.5
      ST_IC  = ^b010 ;  " State #3 of AHPL Description of Figure 12.5
      ST_IC  = ^b011 ;  " State #4 of AHPL Description of Figure 12.5
```

```
" Appendix C Con't.

" Input Pin Names

    LCLK      pin  1 ; " Local Bus Clock.
    QValid    pin  2 ; " Asserted by IMS C011 when a byte is valid.
    IAck      pin  3 ; " Asserted by IMS C011 when a byte is accepted.
    !Out_FF   pin  4 ; " Output FIFO Full Flag  (active low).
    !In_EF    pin  5 ; " Input FIFO Empty Flag (active low).
    !LRESET   pin  6 ; " Local bus reset pulse from NBIC.

" Output Pin Names

    QAck      pin  23 ; " Asserted after byte has been written
    IValid    pin  22 ; " Asserted when a byte has been read from FIFO.
    WRITE     pin  21 ; " Write signal used to derive FIFO write pulse.
    READ      pin  20; " Read signal used to derive FIFO read pulse.

" Declare Output for State-Machines

    ST_OUT1,
    ST_OUT0    pin 19,18;
   STATE_OUT = [ST_OUT1, ST_OUT0] ;

    ST_IN1,
    ST_IN0     pin 17,16;
   STATE_IN = [ST_IN1, ST_IN0] ;

  @radix 16;  " Set base to hexadecimal

equations

state_diagram   STATE_OUT   " Output Port Controller IMS C011 Output
                            " Byte transferred from IMS C011 --> Controller

   " Begin AHPL Description of Figure 12.9

   state ST_OA :  WRITE =  OFF;  " Disable write to FIFO
                  QAck   = OFF;  " Disable acknowledgment to IMS C011
                  if ( LRESET ) then ST_OA          " Stay in this state until
                    else                            " the IMS C011 asserts
                      if ( QValid ) then ST_OB      " is QValid signal.
                        else ST_OA ;
```

```
" Appendix C  Con't.

  state ST_OB :  WRITE =  !Out_FF;  " Write = 1 when FIFO full flag is deasserted
                 QAck    = OFF;  " Disable acknowledgment to IMS C011
                  if ( LRESET ) then ST_OA         " Stay in this state if the
                    else                           " FIFO Full Flag is asserted
                      if ( Out_FF ) then ST_OB     " else go to State C
                        else ST_OC ;


  state ST_OC :  WRITE =  OFF;  " Disable Write to FIFO
                 QAck    = ON  ;  " Assert acknowledge signal to IMS C011
                  if ( LRESET ) then ST_OA         " Stay in this State until
                    else                           " the IMS C011 deasserts
                      if ( QValid ) then ST_OC     " its  QValid signal.
                        else ST_OA ;

  state ST_OD :  WRITE =  OFF;  " Disable Write to FIFO
                 QAck    = OFF ;  " Deassert acknowledge signal to IMS C011
                  if ( LRESET ) then ST_OA ;


state_diagram   STATE_IN      " Output Port Controller IMS C011 Input
                               " Byte transferred from Controller --> IMS C011

  state ST_IA  :  READ   =  OFF;  " Disable Read to FIFO
                 IValid   = OFF ;  " Deassert valid byte signal to IMS C011
                   if ( LRESET ) then ST_IA         " Stay in this State until
                     else                           " there is data to process
                       if ( In_EF ) then ST_IA      " in the Input FIFO.
                         else ST_IB ;

  state ST_IB  :  READ   =  ON   ;  " Read byte from Input FIFO
                 IValid   = OFF ;  " Deassert valid byte signal to IMS C011
                   if ( LRESET ) then ST_IA         " Read byte from Input FIFO
                     else ST_IC ;                   " and goto ST_IC.

  state ST_IC  :  READ   =  OFF;  " Disable Read to FIFO
                 IValid   = ON   ;  " Assert valid byte signal to IMS C011
                   if ( LRESET ) then ST_IA         "Stay in this State until
                     else                           " the IMS C011 acknowledges
                       if ( !IAck ) then ST_IC      " the byte.
                         else ST_ID ;
```

" Appendix C Con't.

```
  state ST_ID : READ  =  OFF; " Disable Read to FIFO
                IValid  = ON  ; " Assert valid byte signal to IMS C011
                  if ( LRESET ) then ST_IA          " Stay in this State until
                    else                            " until IAck is deasserted
                      if ( IAck ) then ST_ID        " by IMS C011, then goto
                        else                        " either ST_IB or ST_IA.
                          if ( !IAck & !In_EF ) then ST_IB
                            else
                              ( !IAck & In_EF ) then ST_IA ;

test_vectors ( [LCLK, !LRESET, QValid, !Out_FF, IAck, !In_EF ]
            -> [ STATE_OUT, QAck, WRITE, STATE_IN, IValid, READ ]


 [C, 0, 0, 0, 0, 0] -> [ ST_OA, 0, 0, ST_IA, 0, 0 ];
 [C, 1, 0, 0, 0, 0] -> [ ST_OA, 0, 0, ST_IA, 0, 0 ];
 [C, 1, 0, 0, 0, 0] -> [ ST_OA, 0, 0, ST_IA, 0, 0 ];
 [C, 1, 1, 0, 0, 0] -> [ ST_OB, 0, 0, ST_IA, 0, 0 ];
 [C, 1, 1, 0, 0, 0] -> [ ST_OB, 0, 0, ST_IA, 0, 0 ];
 [C, 1, 1, 0, 0, 0] -> [ ST_OB, 0, 0, ST_IA, 0, 0 ];
 [L, 1, 1, 1, 0, 0] -> [ ST_OB, 0, 1, ST_IA, 0, 0 ];
 [C, 1, 1, 1, 0, 0] -> [ ST_OC, 1, 0, ST_IA, 0, 0 ];
 [C, 1, 1, 1, 0, 0] -> [ ST_OC, 1, 0, ST_IA, 0, 0 ];
 [C, 1, 1, 1, 0, 0] -> [ ST_OC, 1, 0, ST_IA, 0, 0 ];
 [C, 1, 1, 1, 0, 0] -> [ ST_OC, 1, 0, ST_IA, 0, 0 ];
 [C, 1, 0, 1, 0, 0] -> [ ST_OA, 0, 0, ST_IA, 0, 0 ];
 [C, 1, 0, 1, 0, 1] -> [ ST_OA, 0, 0, ST_IB, 0, 1 ];
 [L, 1, 0, 1, 0, 1] -> [ ST_OA, 0, 0, ST_IB, 0, 1 ];
 [C, 1, 0, 1, 0, 1] -> [ ST_OA, 0, 0, ST_IC, 1, 0 ];
 [C, 1, 0, 1, 0, 1] -> [ ST_OA, 0, 0, ST_IC, 1, 0 ];
 [C, 1, 0, 1, 1, 1] -> [ ST_OA, 0, 0, ST_ID, 1, 0 ];
 [C, 1, 0, 1, 1, 1] -> [ ST_OA, 0, 0, ST_ID, 1, 0 ];
 [C, 1, 0, 1, 0, 1] -> [ ST_OA, 0, 0, ST_IB, 0, 1 ];


end output_controller;
```

REFERENCES

Catherine Mikkelsen, "NeXTbus Interface Chip Preliminary Specification", NeXT Computer Inc., published October 11, 1989.

Catherine Mikkelsen, " NeXTbus Specifications Final Draft", NeXT Computer Inc., Reorder Product #N6010, published October 11, 1989.

Cypress Semiconductor, BiCMOS/CMOS Databook, Cypress Semiconductor, San Jose California, published March 1, 1990.

Harold S. Stone, Microcomputer Interfacing, Addison-Wesley Publishing Company, February 1983, Reading, Massachusetts.

Hill J. F. & Peterson R. G., Introduction to Switching Theory & Logical Design 3ed. John Wiley & Sons, Inc., New York, New York, 1981.

Hill J. F. & Peterson R. G., Digital Systems: Hardware Organization and Design 3ed. John Wiley & Sons, Inc., New York, New York 1987.

Inmos, The Transputer Databook 2ed. , Redwood Burn LTD Publishing Company, Trowbridge, 1989.

Inmos, The Transputer Development and iq systems Databook 1ed., Redwood Burn LTD Publishing Company, Trowbridge, 1989.

W. D. Peterson, The VMEbus Handbook: A User's Guide to the IEEE 1014 and IEC 821 Microcomputer Bus, VITA Publications, VMEbus International Trade Association; Scottsdale, Arizona, 1983.

Appendix III

Research Paper
on
A Benchmark
for
Evaluating Computer
Performance
in
Flight Simulator
Applications

A PORTABLE BENCHMARK FOR SIMULATOR PROCESSORS

Thomas L. Clarke
Institute for Simulation and Training
University of Central Florida
Orlando, Florida

ABSTRACT

Training simulator software makes unique demands on computing hardware. The synchronous, interrupt-driven simulator computational environment is not well approximated by standard benchmarks. FORTRAN code from an T-2C trainer supplied by the Naval Training Systems Center (NTSC) has been used as the basis for a portable benchmark. The code was converted to the C language which is generally the first high-level language available on new processors. The C-language training simulator benchmark has been used to evaluate a variety of RISC (Reduced Instruction Set Computer) and CISC (Complex Instruction Set Computer) architectures. These include the Sun SPARC, the MIPS R2000, the Motorola 88000, the IBM RS/6000, the Motorola 68030 and the Intel 80386. Results are about as would be expected on the basis of standard benchmarks with the exception of the Motorola 88000, whose performance is so fast as to raise skepticism.

INTRODUCTION

In the last few years a new type of microprocessor has gained prominence. These new microprocessors are based on the reduced instruction set design philosophy. Controversial at first, computers based on RISC (Reduced Instruction Set Computer) processors have become very successful as technical workstations. Conventional CISC (Complex Instruction Set Computers) continue to remain dominant in many arenas, however.

Because of the long life cycle of the embedded computer in training simulators, it is especially important to assess the value and longevity of the RISC processor. Is the RISC processor just a "flash in the pan" that will vanish five years from now in favor of a new generation of CISC processors, or do RISC processors represent an enduring change in the art of computer design? In addition, do the RISC features that serve so well in workstations offer improved performance in the simulator environment?

CISC processors are in most cases "code museums" which continue to be used because of the large body of available CISC software [1]. RISC practitioners zealously push the virtues of the RISC simplicity. In Hal Hardenbergh's words, there is a tendency for RISC processors to become "religious artifacts". RISC advocates sometimes simplify their machines to the detriment of performance.

This paper is a report on research designed to assess the utility and performance of RISC processors in the simulator environment.

ARCHITECTURAL ISSUES

RISC (Reduced Instruction Set Computer) processors offer improved performance relative to conventional CISC (Complex Instruction Set Computer) architectures by trading complexity for speed. The simplified instruction set of a RISC processor facilitates the use of techniques such as instruction pipelining and hardwired control of execution that increase processor speed. Pipelining breaks instructions into parts such as load, execute and store whose execution can be overlapped. Hardwired control insures that each part of an instruction completed within one clock cycle, so that overall one instruction is executed per clock cycle. In contrast a CISC generally takes many cycles to complete one instruction. Since the clock rate is limited by the speed of the transistors produced by the given integrated circuit (IC) technology, a RISC will be able to execute more instructions per second than a CISC.

The performance of a RISC is not totally without cost, however. Processor clock rates are generally faster than memory speeds so that a faster, more expensive cache or buffer memory must be inserted between the processor and the slower, less expensive main memory. The cache memory stores the most recently accessed data and instructions so that when the processor is executing a loop (a very common circumstance), the effective memory speed is determined by the cache.

Except for cost, the cache memory poses no problem for most types of computer processing. In simulator and other real-time systems, the cache memory can cause determinism problems. Due to the interrupt driven nature of the simulation environment, the necessary instructions may not be in the cache when an interrupt occurs. If not, the RISC processor must slow to system memory speed. The interrupt service time will then appear to be random since it depends on what other tasks the processor is executing at the time.

The determinism problem can be avoided by not using a cache, but RISC processor performance then drops to the level of CISC. As Figure 1 suggests, there is a performance continuum depending on the "granularity" of the program. Granularity is a measure of the size of the typical fragment of code being executed. Tight loops have small granularity, whereas code driven by external events as in a simulator would tend to have a large granularity. The code executed in response to an external interrupt would on average be far away from the currently executing task.

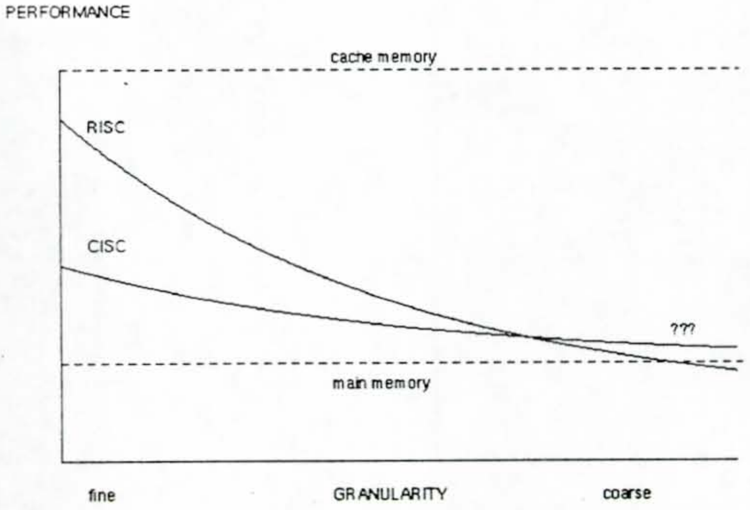


Figure 1. Schematic view of memory access limited performance of CISC and RISC processors as a function of problem granularity.

It is a main goal of this research to determine where on the performance curves typical simulators fall. If they fall to the left or small granularity side, then RISC processors have a clear advantage. If they have large granularity to the right then it a toss up as to whether RISC is better than CISC.

In addition to issues related to cache memory, there are several other architectural issues that are important in determining the RISC versus CISC performance trade-off. Stephen Furber [2] gives an excellent discussion of the architectures of available RISC microprocessors. Briefly these are:

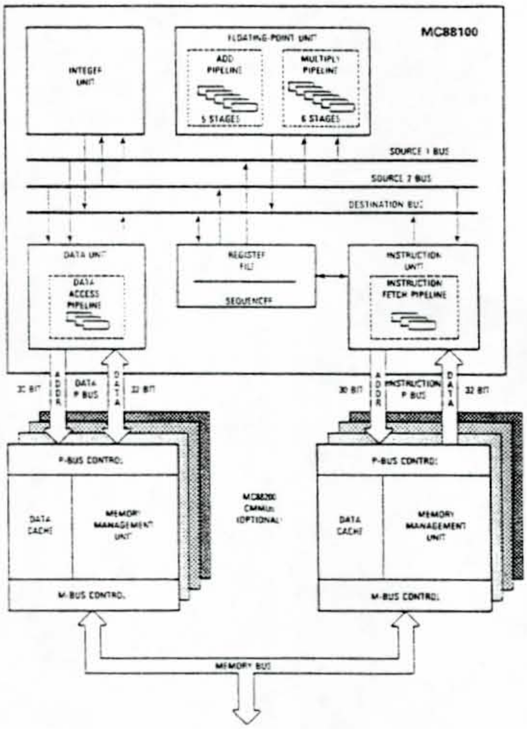


Figure 2. Block diagram of Motorola 88000, showing major features of RISC architecture. Figure taken from [3].

1. **Register Set Size**. A smaller number of registers permits a greater variety of instructions to be encoded in the instruction word because fewer bits are needed to specify the register. This may permit extra instructions better tailored to the simulator workload and resulting in increased performance. More registers, on the other hand permit the processor to switch contexts and respond to interrupts more rapidly. When an interrupt occurs, the processor can simply switch to using an alternate set of registers rather than having to save the contents of a smaller register set. This rapid response may outweigh the penalty of the circumlocutions required by a smaller instruction set.

2. **Location of Floating Point Unit.** Some RISC processors incorporate a closely integrated on chip floating-point arithmetic unit (FPU). In other designs, the FPU is a more loosely coupled off-chip unit. There are speed penalties associated with moving off-chip, but moving the FPU off-chip frees silicon for use as additional registers or as on-chip high speed memory (RAM). Again, the best compromise for the simulation environment is not clear.

3. **Bus-Structure.** Some processors use the conventional von Neumann single bus which combines data and instruction transmission. Other processors use the Harvard architecture in which a separate bus is used for instructions and for data. Separate buses permit a higher data-rate, but have corresponding costs in silicon area and hence other capabilities.

4. **Special Functions.** Some RISC processors feature the capability of expansion. The Motorola 88000 in particular is designed to permit the incorporation of user designed special-function units (SFUs). These SFUs are intimately coupled to the processor in the same way as the FPU and offer the option of incorporating special simulator specific functions in the hardware.

5. **Network Support**. Adequate performance is presently obtained from simulators only by combining several computers into a multi-processor network. The same multiple processor configuration will be needed with RISC technology. It is thus important to evaluate the networking capabilities of the various RISC processors. One of the strengths of the Inmos transputer is the ease with which it interconnects into networks. Customized network interfaces

may be a vital component of the custom VLSI RISC processor developed during later years of this research.

6. **High Level Language (Ada)Support**. Simulator software is broken down into Computer Software Components (CSC); each CSC controls one function of the simulation, e.g. landing gear, engine, etc. In the Ada language, each CSC might correspond to an Ada task. During an update cycle of the simulator, a CISC computer must be able to handle all the CSCs as interrupts occur; this may result in excessive overhead due to context switching. A RISC could operate by handling each CSC completely before beginning the next, avoiding much overhead. The RISC processor thus might be a much more efficient Ada platform.

BENCHMARK DEVELOPMENT

Comparing performance between widely different processor architectures is a common problem. The general computing world has developed a variety of benchmark programs whose performance on a target machine is widely accepted as a measure of the performance to be expected from that machine. Two of the most commonly used are the dhrystone benchmark and the whetstone benchmark. Omri Serlin [3] discusses both of these benchmarks in detail.

The dhrystone benchmark is designed to measure the performance of a processor in general computing applications. It is a synthetic program that exercises the integer, character and control instructions of a machine.

The Whetstone benchmark is designed to test a machines ability to perform numerical calculations. The Whetstone benchmark was originally coded in Ada but has been translated to FORTRAN and C. The Whetstone benchmark consists of multiple loops containing matrix, and transcendental mathematics operations. Many of the calculations are embedded in procedure (subroutine or function for FORTRAN and C) calls to avoid optimization. In a fairly simple benchmark, the calculations are necessarily repetitive and trivial and a optimizing compiler sometimes will optimized them out of existence.

No benchmark is designed specifically to measure performance of a processor in the simulator environment. As part of a Florida High Technology and Industry Council sponsored project, the Institute for Simulation and Training (IST) has taken on the development of such a benchmark.

This new benchmark must exercise the same features of a processor that are exercised in a simulator. At the same time the benchmark must be reasonably portable between various processors. This two requirements conflict, as it is impossible to exercise the hardware interrupt capabilities of a processor with a portable benchmark. The details of interrupt servicing are intimately tied to each specific processor.

An approach was taken that has proven useful at the Naval Training Systems Center (NTSC). In developing a benchmark for in-house use, NTSC took code from a T-2C training simulator and modified it to remove the machine-dependent interrupt-driven portions. The interrupt driven portion was replaced with a main routine that executes a main control loop which corresponds to the simulator frame time. Within each iteration of the loop the various service routines are called on a quasi-random basis. This approach has the advantage that the operations performed by the processor running the benchmark will be identical to the operations performed in the simulator. While the absolute performance will differ between the simulator and the benchmark, relative performance measures between processors should be the same for simulator and benchmark.

Bill Rowan of NTSC kindly made the FORTRAN source code for benchmark available to IST. To further increase the portability of the benchmark, it was decided to translate from FORTRAN to C. The C language is generally the first high-level language available on a new processor, so that benchmarking can be carried out at an early date. This decision has proven a good one at it has made it possible to run the benchmark on a wider range of machines.

The FORTRAN code was converted to C using the commercial FOR_C translator from Cobalt Blue, Inc. FOR_C was very useful and took care of most of the mechanics of translation such as converting SUBROUTINES to functions and the like. Nevertheless, a fair amount of hand coding had to be done as the code dates from the mid-70s and was originally used many techniques to adapt it to small host computers such as the PDP 11/45. The most troublesome technique was the use of EQUIVALENCE to establish correspondence between data arrays with two different names. These techniques are discouraged, if not forbidden, under modern software development guidelines.

The original program structure, consisting of some 117 separately compiled program modules, was retained. While many modules are short and could have been combined into a single C function, this would have opened the danger of having some calculations optimized out of existence. Also, as noted above, many small modules separately compiled and linked together and then called quasi-randomly is about as close to the true interrupt driven simulator environment as possible as can be gotten in a portable benchmark.

The control program was modified to automatically fly four scenarios that have been used at NTSC for benchmarking. These scenarios are basically level flight with various step or impulse inputs to the control surfaces. At the time of writing the benchmark compiles error free and runs with little trouble on a variety of machines. It, however, crashes before the complete test run is complete, probably due to some lingering FORTRAN to C conversion problems. After the crash, the simulator resets and continues flying. This behavior should not influence the relative utility of the benchmark timings. The computations needed to crash and reset are also representative of simulator computational loads. Before the benchmark is released to industry, this final bug will be exterminated, however.

PERFORMANCE MEASUREMENTS

Before discsussing the results, the units of the simulator benchmark need to be named. No euphonious combination involving *stone* came to mind, but then the alleged brick-like aerodynamics of certain aircraft suggested:

> **brick•bat** \'brik-,bat\ *n*
> [*brick* + *bat* (lump, fragment)]
> (1563)
> **1:** a fragment of a hard material (as a brick); *esp*: one used as a missile
> **2:** an uncomplimentary remark

Brickbat has the requisite two syllables and the missile-like connotation of definition 1 is appropriate. In addition, definition two is relevant to those processors that don't do well on the benchmark.

The benchmark was tested on all the readily available workstations at IST. These machines included two conventional CISC machines and four RISC machines. Results are shown in the table along with published whetstone and dhrystone ratings. Brickbats will be defined as 1000 divided by the benchmark execution time.

PERFORMANCE TABLE

Processor	dhrystones	whetstones	brickbats
RS/6000	60700	27250	27.4
88000	34000	6676	108.7
SPARC	21700	5184	11.6
MIPS	18400	5757	9.2
68030	5720	1477	4.5
80386	3000	546	3.9

The timings were performed by running the brickbat benchmark under control of the UNIX /bin/time program which reports the amount of time actually used by the user program while executing.

Beginning with the slowest, the first CISC machine was a Hewlett Packard QS/16 using the Intel 80386 processor. The 80386 is a standard 32 bit "code museum" that derives from the PC tradition. It ran at 16 MHz with 100 nsec memory.

The second-slowest CISC machine was a NeXT workstation using a Motorola 68030 processor. It ran at 25 MHz with 100 nsec memory. Apparently the memory speed prevented the 68030 from benefiting much from its faster clock rate.

The four RISC workstations were a SG 4D70GT using the MIPS R2000 processor running at 16.7 MHz, a SUN Sparcstation using the SPARC processor at 20 MHz,a Data General Aviion using the Motorola 88000 at 16 MHz, and an IBM RS/6000 workstation using IBM's RISC chipset running at 25 MHz. Little technical information is available about these machines beyond the clock rate.

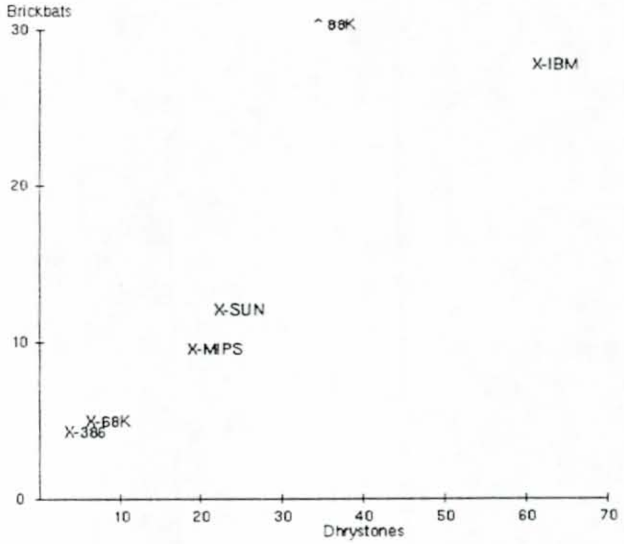


Figure 3. Comparison of dhrystone benchmark results with Brickbat results.

The performance of the various processors ranged from a low of 3.9 brickbats (256 seconds execution time) for the 80386 to a high of 108.7 brickbats (9.2 seconds) for the 88000. As discussed below, the figure for the 88000 seems anomalously low, so that the real range was probably up to the 27.4 brickbat (36.5 seconds) value of the IBM RS/6000.

Both the dhrystone and the whetstone bench marks were good predictors of the brickbat value. Figure 3 plots brickbats versus dhrystones. When the Motorola 88000 is excluded (off scale in figure 3), the correlation between either dhrystones or whetstones and brickbats is greater than 90%. This is not surprising since all processors tested include floating point hardware so that there is a good correlation between dhrystones and whetstones. Figure 4 plots brickbats versus whetstones.

The results for the Motorola 88000 seem anomalous. Its performance is better by a factor of nearly eight compared to what would be expected on the basis of its dhrystone or whetstone rating. No reasons for the anomaly have been identified. The flying time of the benchmark was varied to perhaps isolate a time anomaly in the Aviion, but the results tracked as if the 88000 is truly very fast.

There only feature of the 88000 architecture that might account for this speed is the Harvard style of memory access. As figure 2 shows, the 88000 has two complete paths to cache memory: one for data and one for instructions. It is difficult to see how this dual bus architecture would account for more than a factor of two in performance, however.

Possibly the combination of Harvard architecture and a separate autonomous floating point unit enable the routine calling and return code to effectively overlap calculations. Since the brickbat benchmark consists of many calls to small routines which are calculation intensive, this may result in a large speed-up factor. The IBM RS/6000 architecture also features multiple paths from cache (DCU in figure 5) to the instruction processor (ICU) and the floating (FXU) and fixed (FXU) arithmetic units. The IBM FPU is described as tightly coupled, however, so perhaps something about the simulator instruction mix favors the 88000.

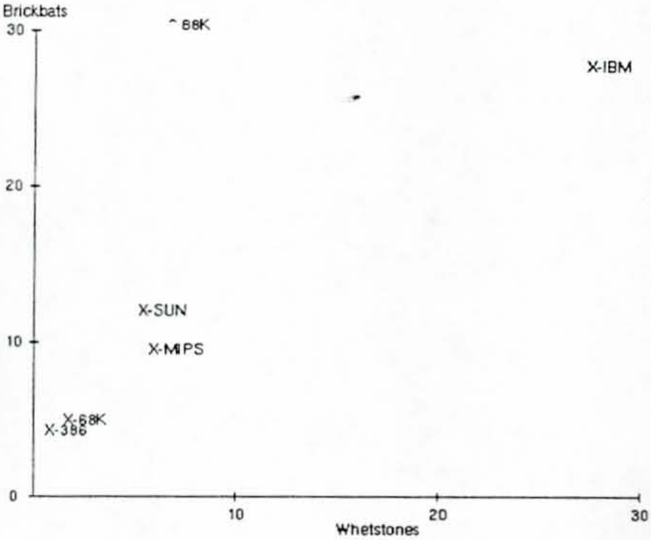


Figure 4. Comparison of whetstone benchmark results with Brickbat results.

CONCLUSIONS

Training simulator software makes unique demands on computing hardware. The synchronous, interrupt-driven simulator computational environment is not well approximated by standard benchmarks. The brickbat benchmark reported here which is based on code from a T-2C trainer should provide an approximation of that environment.

The C-language brickbat training simulator benchmark has been used to evaluate a variety of RISC (Reduced Instruction Set Computers) and CISC (Complex Instruction Set Computers) architectures. These include the Sun SPARC, the MIPS R2000, the Motorola 88000, the IBM RS/6000, the Motorola 68030 and the Intel 80386. Results are as expected on the basis of standard benchmarks with the exception of the Motorola 88000, which seems too fast to be true.

While the Motorola 88000 thus seems to be the best processor tested for simulators. The excessive amount of its performance increment, raises doubts as to whether this performance would be realized in an actual simulator.

Many interesting processors, such as Intel's i960 and 80486, Motorola's 68040 , AMD's 29000, etc., have not been tested. Some such as the Inmos transputer and the intel i860 will be tested in the near future. In addition this benchmark will be made available to industry and should prove extremely useful in evaluating computers for use in simulators. The benchmark will be offered to the Systems Performance Evaluation Cooperative (SPEC) for inclusion in their suite of benchmarks.

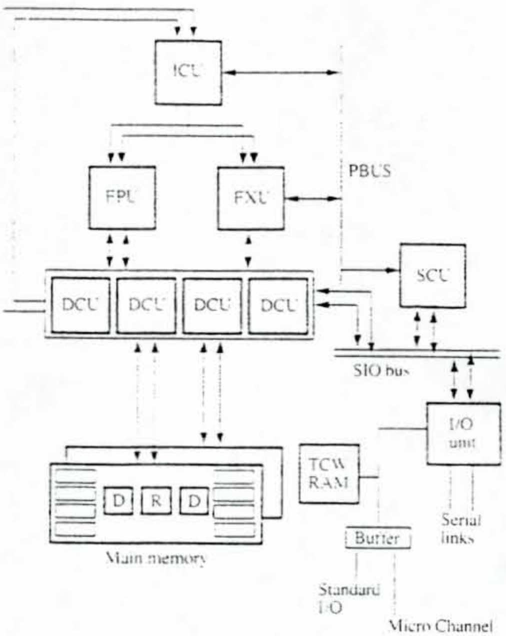


Figure 5. IBM RS/6000 Architecture. From Bakoglu *et al* [5]

REFERENCES

[1] Hardenbergh, Hal. "Religious Artifacts and Code Museums", *Dr. Dobb's Journal, 15*, 2, 13-131. 1990.

[2]Furber, Stephen B. *VLSI RISC Architecture and Organization, Marcel Dekker,* New York, 1989.

[3]Motorola. *MC88100 RISC Microprocessor User's Manual*, Prentice Hall, New York, 1990.

[4]Serlin, Omri. "MIPS, Dhrystones and Other Tales" in *Reduced Instruction Set Computers,* edited by William Stallings, IEEE Press, Washington, 1986.

[5] Bakoglu, H.B., G.G. Grohoski, and R.K. Montoye, "The IBM RISC System/6000 processor: Hardware Overview", *IBM J. Res and Dev, 34*, 1, 12-22.

ACKNOWLEDGEMENTS

Thanks to Bill Rowan of NTSC for providing the flight simulator code.

This research was funded by the Florida High Technology and Industry Council (FHTIC) as part of a project to evaluate the utility of RISC processors in simulators.

ABOUT THE AUTHOR

Thomas Clarke is a Senior Scientist at the University of Central Florida's Institute for Simulation and Training where he serves as Principal Investigator on a variety of simulation and training related projects. He has a doctorate in applied mathematics from the University of Miami.

Appendix IV

Technical Portion
of
Proposal
to DARPA
Based on
Technology Developed
Under FHTIC
Sponsorship

Volume I - Technical

DARPA BAA #90-15
(Stategic Computing Program)

Optimal Virtual World Displays

Principal Investigator:

Thomas L. Clarke
Institute for Simulation and Training

University of Central Florida
12424 Research Parkway, Suite 300
Orlando, FL 32826
(407) 658-5030
clarke@ucf-cs.ucf.edu (internet)
TCLARKE@UCF1VM (bitnet)

Administrative POC:

Irene Martin
Division of Sponsored Research
University of Central Florida
PO Box 25000, Orlando, FL 32816
(407) 823-3778

Table of Contents

List of Figures

B. SUMMARY of INNOVATIVE CLAIMS

Small low-power visual displays for use with embedded microsystems applications such as virtual reality require finesse rather than the brute force approaches used for conventional visualization displays. Research at IST funded by the State of Florida has identified the utility of differential-geometric concepts in algorithms for generating graphics that are optimally matched to the characteristics of the human visual system.

Differential geometric techniques together with distorting optics will be used to optimally utilize scarce embedded microsystem computational resources. In principle, two million properly distributed pixels are sufficient to feed the million or so fibers in the optic nerve, producing a perfect display. In practice, a 512 by 512 pixel display with proper optics and head position feedback will produce a convincing illusion of reality.

Significant optical size and weight savings can be realized by designing the optics and the distorting algorithms as an integrated system. For example, chromatic aberrations can be corrected by sizing the three images in a color display to match the optics. Conversely, the proper choice of optics can result in substantial savings of computational and display hardware. Distortion can be produced easily with simple optics saving the need for display pixels and computation cycles at the edge of the field of view.

The display system to be developed at IST will have variable pixel sizes across the image. This vari-pixel display will use differential geometry as the mathematical foundation of its image transformation algorithms. Recent work has made it clear that Lie transformation groups are the most appropriate mathematics for visual transformations. This is no accident; as in physics, transformation group symmetries provide the best expression for models of physical reality.

The display system would consist of image warping algorithms together with special optics that would give a variable resolution image matched to the eye's characteristics. As mentioned above and detailed in the Technical Rationale, designing the optics in conjunction with the algorithm can lead to simplifications in the optics, saving cost and weight.

The display system developed as part of this research will be truly optimal in that it will require the least number of pixels possible for a wide field of view, full-resolution image. The demonstration system will utilize graphic workstation hardware and custom designed optics to yield throughput equivalent to tens of thousands of polygons per second. Adaptability of the vari-pixel concept to higher performance systems is guaranteed by the local nature of the Lie transformation groups used. Their locality ensures that the warping algorithms can be adapted to the special purpose VLSI parallel processing hardware required for the highest performance.

Applications that will benefit vari-pixel display technology include training, telepresence, virtual reality, and scientific visualization.

C. DELIVERABLES

Year 1:

1. Reports detailing the applicability of group-theoretic transforms to image generation will be prepared.

During the first year, theoretical and experimental studies supported by computer modeling will be used to establish an appropriate design and architecture for the image generation system. The design ideas will be testedusing off-the-shelf computer hardware (e.g. Amiga 3000) and "Sony Watchman" type displays with bread-board optics.

These reports will cover the background concepts of group theory and differential geometry in sufficient detail for someone skilled in image display but unfamiliar with the theory to be able to make use of the research.

2. Copies of software developed to test group-theory based image generation concepts will be delivered.

The software will be programmed in the C language, and runon the computer hardware which generates displays on the "Watchmen". These displays will be viewed through the bread-board optics.

Year 2:

1. Reports detailing the design specifications of a high performance vari-pixel system using workstation-class graphics hardware, state of the art displays and custom optics will be prepared.

NeXT has announced a color graphics coprocessor for the NeXT using the Intel i860 processor. This system (or a system of equivalent performance) will be used to drive the display. Performance enhancement by special purpose co-processors will be considered. Possible co-processors include the Motorola 56001 DSP in the NeXT or arrays of Inmos transputers . The display will be state-of-the art compatible with use as an eye-phone; helmet-mounted-display quality CRTs and fiber optic displays will be considered.

2. Test results detailing progress in implementing the high performance vari-pixel display will be submitted.

Year 3:

1. Reports detailing the high-performance vari-pixel display will be completed. These reports will contain sufficient detail for the system to be reproduced by someone skilled in the art.

2. Detailed software and hardware designs for the vari-pixel display will be submitted.

A number of refereed papers will also result from this research effort.

D. SCHEDULE and MILESTONES

This project is inherently a multi-year project. After a year of theoretical and experimental groundwork, two years will be needed to develop a working prototype of an optimal image display system.

The first year is devoted to theoretical and experimental studies of how best to apply group-theoretical ideas to image display. The broad outlines are clear, as can be seen in the Technical Rationale section, but there are many details that need to be worked out before the prototype is implemented. The budget for the first year will be $98,851.

During the second year, implementation of the display system will begin in earnest. The system software will be coded, and the prototype hardware assembled. The system will be debugged and preliminary testing done. The budget for the second year will be $146,988.

In the third year, the group-theoretic display will be tested, refined and made ready for transition to general use. The budget for the third year will also be $115,092K.

Total project cost will be $360,931.

```
Month                         1  2  3  4  5  6  7  8  9  10 11 12

Year 1

        Report                                                        ^
        Theory                    |--------------------|
        Optical Design                        |-------------------|
        Experimentation                                 |---------|

Year 2

        Report                                                        ^
        Coding                    |-----------------------|
        Construction              |-------------  --------|
        Testing                                                |--------|

Year 3

        Report                                                       ^
        Testing                  |----------------|
        Refinement                         |---------------|
        Implementation                                     |---------------|
```

E. PROPRIETARY INFORMATION

There is no proprietary information needed to support this research.

Any inventions or patentable software that result from this project would be subject to applicable Florida State University System and Federal disclosure regulations.

F. STATEMENT OF WORK

The Institute for Simulation and Training (IST) of the University of Central Florida (UCF) will conduct research designed to develop a novel, high-performance display system (vari-pixel display henceforth) in response to DARPA BAA 90-17 for research into the field of embedded microsystems.

This research encompasses the following tasks:

Year 1:

Conduct theoretical and experimental studies supported by computer modeling to establish an appropriate design and architecture for the image generation system. The design ideas will be tested using off-the-shelf computer hardware such as Amiga 3000 and "Watchman" type displays with bread-board optics.

Prepare a report on the results of these studies including sufficient detail on the background concepts of group theory and differential geometry that someone skilled in image display but unfamiliar with the theory to be able to make use of the research.

Software to test group-theory based image generation concepts. This software will be written in the C language, and will run on the computer hardware which generates displays on the "Watchmen". These displays will be viewable through bread-board optics. Copies of this software will be be made available to the sponsor.

Year 2:

Produce a detailed design for a vari-pixel display with workstation-class performance, state-of-the-art resolution and custom optics. Performance will be equal to that expected for the NeXTdimension color processor using the Intel i860 processor. The display will have the highest state-of-the art resolution compatible with use as an eye-phone; helmet-mounted-display quality CRTs and fiber optic displays will be considered. Optics will be of a solid and robust design using quality components.

The software developed to implement the pre-distortion will be documented so that it is useful to other developers.

Details of the design and results of testing the components will be reported to the sponsor.

Year 3:

The high-performance vari-pixel display will be integrated and fully tested; its performance will be evaluated as a display for virtual reality, as a workstation visualization display, as a display for telepresnece and as a display for training.

The detailed design of the vari-pixel display will be delivered to the sponsor. Recommendations for future directions of research and development will be made to the sponsor. In addition, copies of papers and conference proceedings resulting from the work will be delivered to the sponsor.

H. RESULTS, PRODUCTS, and TRANSFERABLE TECHNOLOGY

The vari-pixel display technology to be developed as part of this research will be of potential benefit to many users of computer graphics displays. The following paragraphs describe how the technology will impact various users.

The Virtual Reality Explorer

A user would find the vari-pixel displays developed under this project to be a revelation. Upon donning the phones, the user would find a vast difference in performance compared to current eye-phones. The visible field-of-view is much larger than that in current eye-phones; at the same time, the image appears to be sharper than any currently available. The weight of the eye-phones is also reduced and they are more comfortable to wear. Processing response delay artifacts are reduced compared to other visual systems of comparable apparent resolution and field of view.

On the whole, the illusion of virtual reality is much stronger.

The Programmer

The software tools provided with the vari-pixel display permit the use of many sources of visualization data. Not only are polygonal data bases usable, but the use of Lie group techniques permits the use of photographic data as well.

Efficient use is made of advanced VLSI co-processors so that the host computer is not overly burdened by graphics tasks.

Advanced Application Developers

The advanced developer finds the technology to be ideal for incorporation into portable information systems. The algorithms are a good match to the capabilities of VLSI co-processors.

Available pixels are deployed in a manner designed to match human visual system characteristics; this minimizes overall system cost for given performance level.

The Trainee

A trainee using vari-pixel displays would experience a much more realistic simulation of the training environment. The wider field-of-view would contain more of the cues used in actual performance. The trainee's peripheral vision would be exercised to an extent not possible with current technology; this would avoid false training in which the trainee might learn to ignore peripheral cues.

The Teleoperator

The wide field of view of the vari-pixel display would benefit an operator working remotely through telepresence. As with a trainee, peripheral cues are very important to someone working remotely through a data link.

The pre-distortion and optical rectification used in the vari-pixel technique is also a form of data compression. The operator will be able to receive the same amount of visual information over a much narrower bandwidth signal.

Technology Transfer to Other Products

The vari-pixel display is an enabling technology. By providing a relatively low-cost but high resolution and wide field of view display, the vari-pixel display will open up a potentially wide range of applications. The uses listed above are those that are conventionally associated with virtual reality and like technologies. The verisimilitude of the vari-pixel display is likely to suggest other applications.

Various possibilities include:

¤ medical telepresence - like *Fantastic Voyage*

¤ telepresence in manufacturing - multiplying the skilled worker

¤ consumer applications - if the price falls enough.

H. TECHNICAL RATIONALE

The Problem

Virtual reality is a new technology with "potential to radically alter the way many people interact with computers" (VanName and Catchings, 1990). The virtual reality user is provided with a direct sensory input to the computer such as a data gloves and other position sensors. In turn the computer generates a visual scene that the user views through a head mounted display, or eye-phone. The user's interaction with the computer is direct and intuitive and avoids the awkward bottleneck found in other approaches to visualization of data.

Achieving the promise of virtual reality requires that the visual display be realistic enough so that the user can suspend disbelief. This level of reality requires a display that updates rapidly enough to avoid jerky motion as the user moves about in the virtual world. The display must also have high enough spatial resolution to present a realistic appearance free of aliasing and other sampling artifacts. Telepresence makes similar demands on display technology.

Achieving speed and resolution requires large computational resources and exotic display technology. It is the goal of this research to reduce the amount of computation required and to improve the performance of conventional display devices in virtual reality applications.

An ideal virtual reality display would present the user with a full visual hemisphere at the human eye's resolution of one arc minute. To achieve full resolution on this hemispherical display (2π steradians) would require 75 million display elements; this resolution is beyond any technology easily affordable for the foreseeable future.

Updating a 75 million pixel display fast enough to avoid flicker and other motion artifacts is also beyond present capabilities. Even if only a few instructions are required to update each pixel, a 30 Hz update rate would require in excess of 10,000 MIPS. Only experimental supercomputers currently achieve this rate of computation.

The Solution

A solution to the problem of virtual reality displays is suggested by the fact that the human visual system does not have uniform resolution across the field of view. As Figure 1 shows, the resolution in the central or foveal area is an arc minute, but drops fairly rapidly away from the fovea. Sixty degrees, approximately one radian, from the fovea the resolution has dropped to only 20 minutes of arc. Approximating the eye's resolution curve as $e^{3\phi}$ (where ϕ is the angle from the fovea in radians) and integrating gives only 7 million resolution elements over the full hemisphere. That is, the effective area over which the eye has 1 minute arc resolution is .65 steradians or ~2000 square degrees.

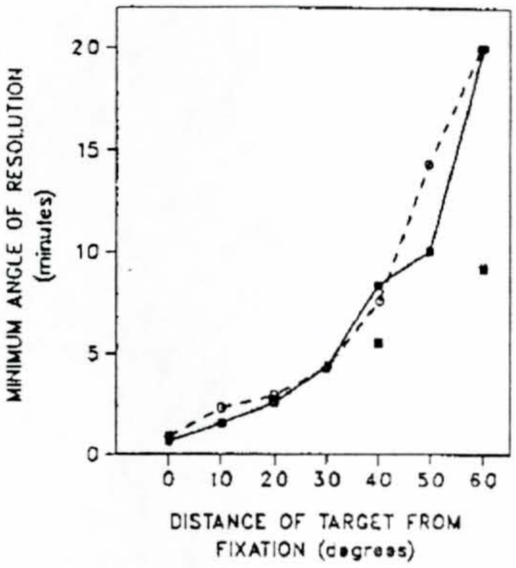


Figure 1. Visual resolution of the human eye (Boff *et al*, 1986).

Seven million pixels is still large for current display technology, but if the resolution is reduced to 2 minutes of arc, the pixel count reduces to about two million pixels. This reduced resolution corresponds to that of a typical workstation screen with a .3 mm color mask viewed at a distance of 600 mm or 24 inches. Two million pixels is within the range of today's large display technology. Small displays such as those used for eye-phones are more limited, but the two million pixel level should be reached in the near future.

Generating a display of 2 million pixels at a 30 Hz update rate must be done efficiently. Even efficient algorithms will require hundreds of MIPS to update the display. This problem can be solved by using VLSI processors together with parallel processing techniques. Single chip processors now achieve tens of MIPS, so that a system with a reasonable degree of parallelism will be able to update the display. The display algorithms, of course, must lend themselves to decomposition into parallel executing sub-tasks.

It is the goal of this project to develop a vari-pixel display that can practically achieve non-uniform distribution of resolution matching the human eye. The vari-pixel display will have two components. The first component is an optical system which magnifies the image of the display device in a controlled non-linear fashion to achieve the desired distribution of pixel resolution. The second component is algorithms for producing appropriately pre-warped images on the display. The pre-distorted image will have correct perspective when viewed through the optics. The details are discussed in the next two sections.

Optical Design

Design Goals

The vari-pixel optics must meet several design diverse goals. They must provide non-linear magnification approximating of the inverse of the human visual acuity curve (Figure 1). The optics must not introduce any perceptible aberrations into the image. Also, the optics must be light and compact enough for use as an eye-phone worn by the virtual reality user.

Fortunately, distortion is fairly easy to achieve. Consider the distorted hemispherical field of view achieved by the simple optics in the common door peep-hole viewer. Also it is fairly easy to equal the human eye's resolution; the diffraction limited design of the Hubble telescope is not needed. A minute of arc on the optical axis, falling to 20 minutes of arc 60 degrees off-axis can be easily achieved as the following example shows.

A Sample Optical Design

When designing an optical system for direct viewing by the human eye there are a variety of optical effects that prove useful (Clarke, 1983):

¤ Parallel beams of light (infinity focus) refracted by a plane surface are free of all aberrations except distortion and lateral chromatic aberration; think of a window or prism.

¤ A spherical surface with a pupil at the center of curvature has only spherical aberration, longitudinal chromatic aberration, and curvature of field ; this is the design principal of the Schmidt camera.

¤ An optical element located at an image plane introduces only curvature of field; this effect can be used to flatten an otherwise curved field.

The optical system for the vari-pixel display shown in Figure 2 makes use of these optical effects. Starting with the eye, parallel light passes through the plane surface nearest the eye. This surface introduces lateral chromatic aberration, and distortion. Leaving consideration of the lateral chromatic aberration for later, the distortion can be desirable.

To get a feel for the magnitudes involved, consider a system using lucite plastic. The index of refraction is then $\mathbf{n = 1.5}$, and a field of $\mathbf{2A_e = 180^{o}}$ at the eye, is reduced to $\mathbf{2A_o = 82^{o}}$ after passing through the plane surface according to the sine law:

$$\sin A_e = n \sin A_o$$

Locally, the pixels at the edge of the field are stretched according to

$$dA_e/dA_o = n \cos A_o / \cos A_e$$

At a field angle of $2A_e = 160^o$, to avoid the singularity at **180°**, the resolution of the image reduces by a factor of nearly 6 at the field edge. This is a good first approximation to the characteristics of the eye and permits the limited number of pixels available in displays to be stretched over a wider field.

A spherical surface is beyond the plane input surface. The radius of this surface is chosen so that the eye's pupil at apparent distance **nE** is located at its center of curvature. Therefore neglecting thickness, **R ~ nE**. This surface thus focuses upon a surface at distance **R/(n-1) ~ nE/(n-1)** introducing only curvature of field, spherical and longitudinal chromatic aberrations. The seriousness of the spherical and chromatic aberrations depends on the focal length and are negligible if **E** is sized for eye-phones. With a 2 inch (50 mm) diagonal display, **E = 16.7 mm** would be appropriate (focus = diagonal). The longitudinal chromatic aberration would be .67 mm. With a 3 mm eye pupil (f/16) the chromatic blur would be 1.43 minutes of arc. This is comparable to the resolution of the eye, and smaller than the limit set by pixel number of practical displays. Third order aberration formulas (Welford, 1974) show that spherical aberration is likewise unimportant.

Curvature of field can be removed with a plano-concave lens located near the image plane so that it introduces no new aberrations. For a flat field, the Petzval sum must be zero; if this lens is made from the same material as the first, its radius must be **-R**. The flat field requirement can be relaxed considerably since the eye has low resolution at the edge of the field of view. In particular, 6 mm of focus shift can be tolerated at the field edge for the 3 mm exit pupil (f/16) assumed above.

Thus two lenses with proper curvature and location leave only lateral chromatic aberration, and distortion as significant aberrations. Distortion is desirable for the vari-pixel display system, but lateral chromatic aberration is not. At the edge of a **160°** field, the edges of the blue and red (F and C lines) images will be separated by **1.78°**. This is several times the eye's 20 minute field-edge resolution and is unacceptable. Even though the eye is relatively insensitive to color at the edge of the field of view, the colored fringes caused by lateral chromatic aberration will be perceived as a degradation in image sharpness.

For a monochromatic display, the lateral chromatic aberration can be minimized by using light over only a narrow range of wavelengths. For the more desirable case of a full-color display, lateral chromatic aberration can be removed in software. Separately warping each of the red, green and blue display images according to the optical system's color-dependent distortion will result in a perceived image that is free of lateral color. This solution requires extra computation, of course, but this can be minimized as will be seen in the section on algorithms.

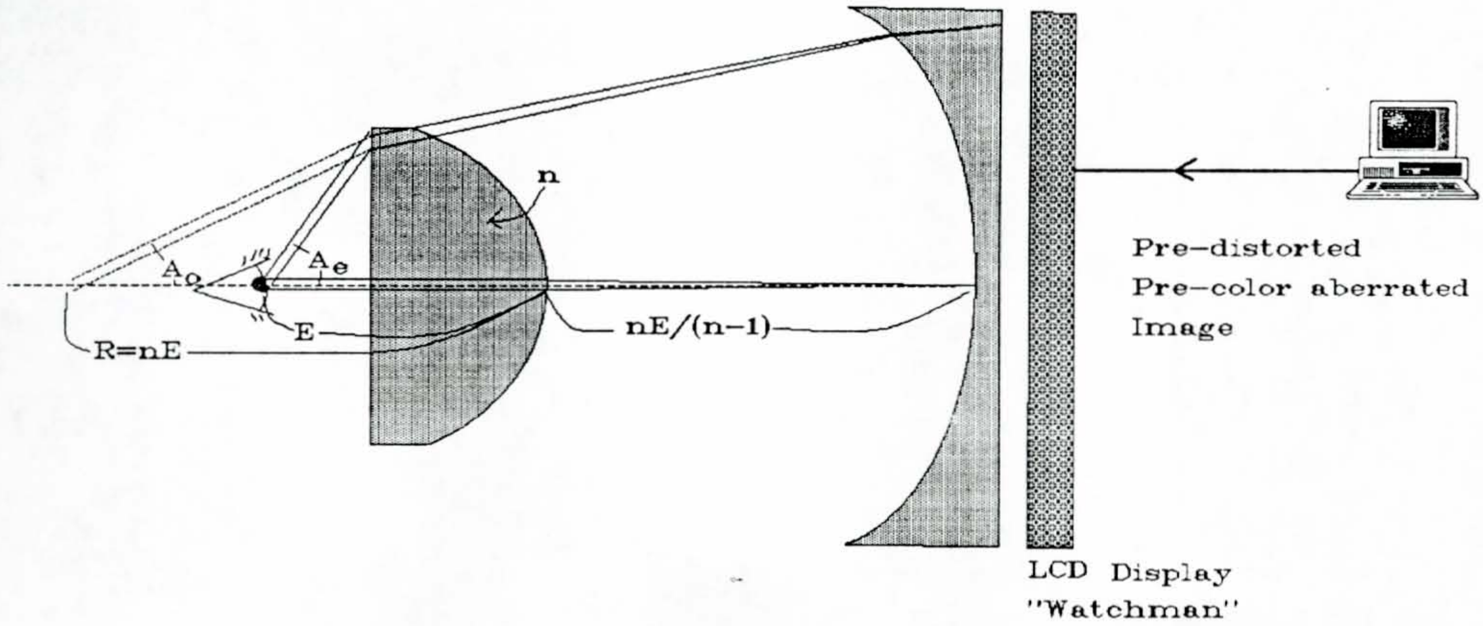


Figure 2. Example of a vari-pixel optical system design.

This sample vari-pixel optical system discussed above distorts a rectangular grid as shown in Figure 3. Pixels at the edges of an image are magnified so that their effective resolution is reduced relative to the center. This variable magnification provides an approximation to the desired curve in Figure 1.

A significant goal of this research is to develop an optimized optical system that closely approximates the desired vari-pixel resolution curve. Several design options will be explored. Rotier (1989) discusses some of the optical possibilities for helmet-mounted displays.

Among the possibilities that will be explored is the use of multiple lenses near the eye so that their distortion contributions are compounded. Mirror optics may have prove advantageous as well since they are inherently free of chromatic aberrations. Half-silvered mirrors can be used to fold the optical path reducing the size of the overall system. Fresnel lenses could useful in reducing the size and weight of the optics. Fresnel lenses are thin plates of optical material embossed with concentric grooves. This results in considerable weight savings relative to a thick spherical lens. The texture of a Fresnel lens should not be a problem as long as the grooves are small compared to the eye's pupil size.

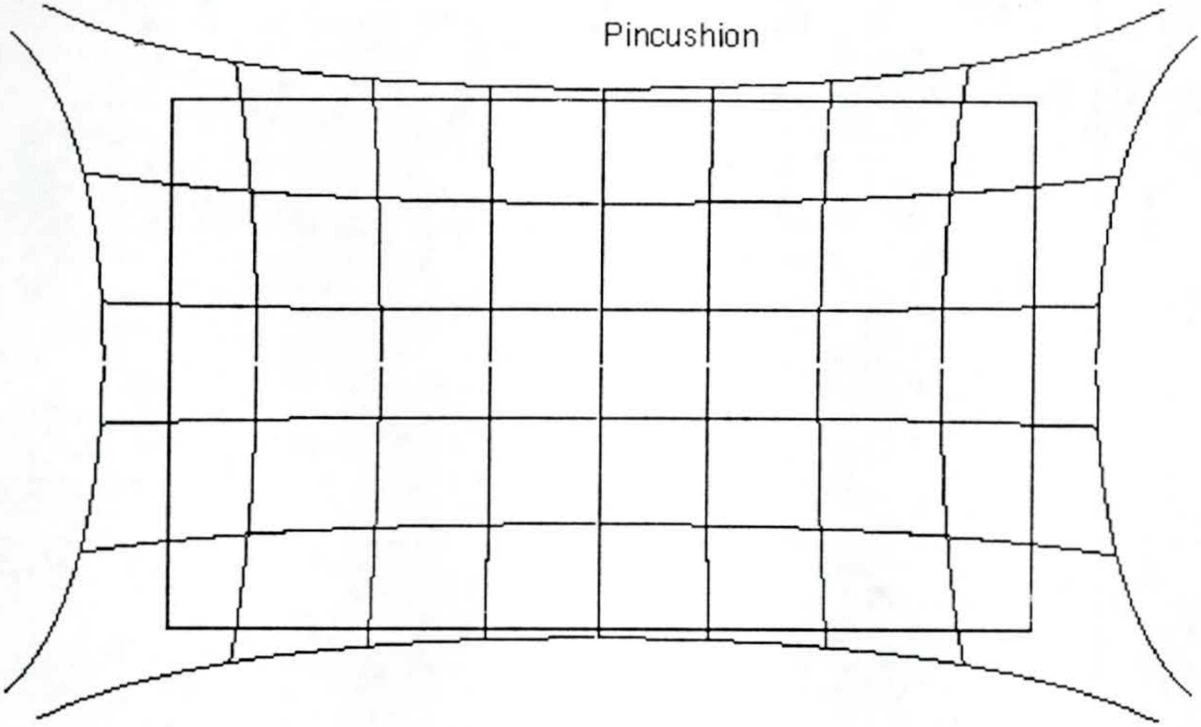


Figure 3. Optical system in Figure 2 warps a uniform grid of straight lines into the hyperbola-like curves in the figure.

Warping Algorithms

In order for the vari-pixel display to provide a view of the virtual world with correct perspective, the image on the display device must be pre-distorted with the complement of the pattern shown in Figure 3. There are many well-know techniques for warping images. Warping algorithms share many techniques with image generation. The following is a brief discussion of some recent work.

In general, there are two broad approaches to generating images: ray tracing and polygonal rendering. Andrew Glassner of Xerox PARC (Palo Alto Research Center) has provided a broad survey of the field. Within Glasser's survey, a particularly valuable work is the stochastic sampling approach which avoids aliasing artifacts by redistributing under-sampled energy across the spectrum. Stochastic casting or rays are a good way to divide effort among an array of VLSI processors. Glassner also discusses various methods of speeding ray-tracing, including the use of vector and parallel architectures. A rather complete ray-tracing bibliography is also included.

Another valuable source is the work edited by Dew, Earnshaw and Heywood (1989). It contains a variety of algorithms oriented toward speeding visualization using parallel computation. In particular, many algorithms for parallel ray-tracing are presented.

The work of Magnenat-Thalmann and Thalmann (1987) is also germane. They present a variety of techniques for dealing with texture that are applicable to the problem of generating images using arrays of processors.

Very recent relevant research has been done by Frederick and Schwartz (1990). They identified the utility of the mathematical technique of conformal mapping for performing the transformations needed in image generation. Their suggestion of the use of abstract mathematical tools suggests the use of the machinery of differential geometry and in particular of Lie groups. In particular, Hoffman (1966 and 1989) has shown the utility of Lie groups in the analysis of vision.

These mathematical tools provide an expressive language for constructing the algorithms needed for vari-pixel imaging. Consider Figure 4 which shows how a rectangular grid must be pre-warped so that the optical distortion of Figure 3 will yield a rectangular grid. The ellipse-like curves in Figure 4 can be naturally handled via differential geometric techniques. Conversely, polygons are warped into regions with curved sides. This presents difficulties for approaches that depend on polygons for rapid image generation.

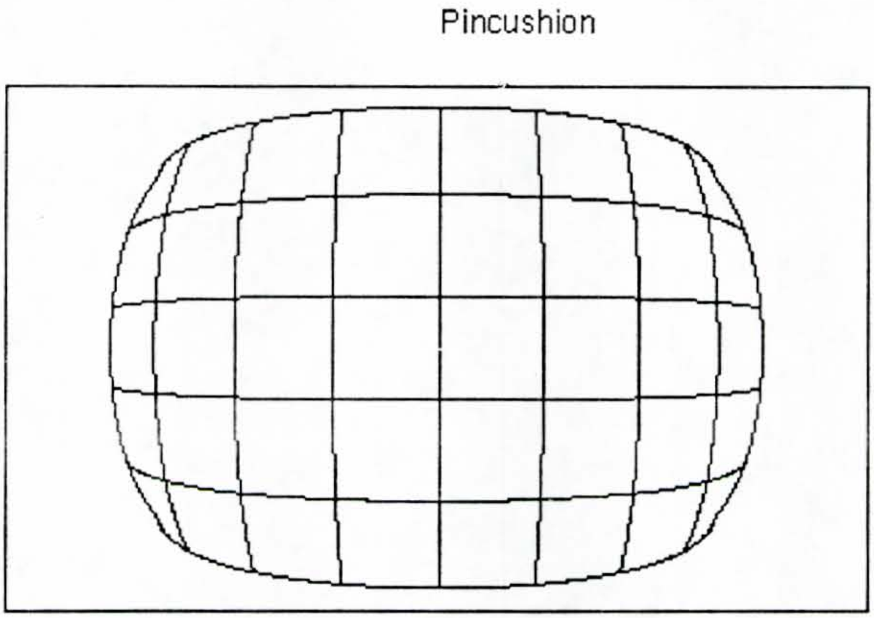


Figure 4. Pre-distortion of a rectangular grid.

A brief summary of Lie theory is needed in order to appreciate in detail its applicability to vision problems. This mathematical material can be bypassed to reach the textual explanation at the end of the section if a qualitative understanding is desired.

The basis of Lie group theory is the concept of a manifold. An m-dimensional manifold is a set $\mathbf{M}$, together with a countable collection of subsets $\mathbf{U}_\alpha \subseteq \mathbf{M}$, called coordinate charts, and 1:1 functions $\chi_\alpha : \mathbf{U}_\alpha \rightarrow \mathbf{V}_\alpha$ onto open sets $\mathbf{V}_\alpha \subseteq \mathbf{R}^m$, called local coordinate maps which satisfy:

(1) The coordinate charts cover **M,** $\cup_\alpha \mathbf{U}_\alpha = M$,

(2) The coordinates are smooth, on $\mathbf{U}_\alpha \cap \mathbf{U}_\beta$,

the composite map

$$\chi_\beta^{-1} \chi_\alpha : \chi_\alpha(\mathbf{U}_\alpha \cap \mathbf{U}_\beta) \to \chi_\beta(\mathbf{U}_\alpha \cap \mathbf{U}_\beta)$$

is $\mathbf{C}^\infty$ (or analytic) , and

(3) The coordinates induce a Hausdorf topology on M,

if $x \in \mathbf{U}_\alpha$, $\mathbf{y} \in \mathbf{U}_\beta$, $\mathbf{x} \neq \mathbf{y}$,

then $\exists$ open sets $\chi_\alpha(\mathbf{x}) \in \mathbf{A}\ \mathbf{V}_\alpha$, and $\chi_\beta(\mathbf{y}) \in \mathbf{B}\ \mathbf{V}_\alpha \ni$

$$\chi_\alpha^{-1}(\mathbf{A}) \cap \chi_\beta^{-1}(\mathbf{B}) = \varnothing.$$

The notion of smoothness of maps between manifolds is needed. If **M** and **N** are $\mathbf{C}^\infty$ manifolds, a map **F:M -> N** is $\mathbf{C}^\infty$ if

$$\forall\ \chi_\alpha : \mathbf{U}_\alpha \to \mathbf{V}_\alpha \subseteq \mathbf{R}^\mathbf{m} \text{ on } \mathbf{M}, \text{ and}$$

$$\forall\ \chi_\beta : \tilde{\mathbf{U}}_\beta \to \tilde{\mathbf{V}}_\beta \subseteq \mathbf{R}^\mathbf{n} \text{ on } \mathbf{N},$$

$$\chi_\beta \mathbf{F}\ \chi_\alpha^{-1} : \mathbf{R}^\mathbf{n} \to \mathbf{R}^\mathbf{m} \text{ is } \mathbf{C}^\infty \text{ on } \chi_\alpha [\mathbf{U}_\alpha \cap \mathbf{F}^{-1}(\tilde{\mathbf{U}}_\beta)].$$

$\mathbf{C}^\infty(M)$ is the set of $\mathbf{C}^\infty$ functions from **M** to **R**.

An r-parameter Lie group can now be defined as a group G which also has the structure of an r-dimensional $\mathbf{C}^\infty$ manifold in such a way that both the group operation

$$\mathbf{m: G \times G \to G,\ m(g,h) = g \cdot h, \quad g,h \in G,}$$

and the inversion

$$\mathbf{i: G \to G,\ i(g) = g^{-1},\ g \in G}$$

are $\mathbf{C}^\infty$ maps between manifolds.

The concept of a vector field requires the definition of the tangent space $\mathbf{T_xM}$ as the m-dimensional vector space determined by the tangent vectors to curves passing through the point **x**.

A tangent vector v is defined geometrically as the equivalence class $\sigma_{\mathbf{x}}$ of curves passing through **x** with all derivatives equal. A tangent vector induces a derivation on $\mathbf{C}^{\infty}(M)$ by

$$\mathbf{vF = dF(\sigma_{x}(t))/dt}\,|_{t=0}$$

Alternatively a tangent vector can be defined algebraically as a derivation, in which cas it can be shown that in general

$$\mathbf{v} = \Sigma_{\mathbf{i}}\, \mathbf{v_i(x)}\, \partial/\partial \mathbf{x}_i$$

for some functions $\mathbf{v_i(x)}$ and stands for the derivation along $\chi_{\alpha}^{-1}\mathbf{(tx_i)}$ where $\mathbf{x_i}$ is the ith coordinate in $\mathbf{R}^{m}$. The tangent bundle is the collection of all tangent spaces

$$\mathbf{TM} = \bigcup_{x \in M} \mathbf{T_x M}$$

the **TM** is a manifold of dimension **2n**.

Finally, a vector field v on M is a smooth map from $\mathbf{v(x)} \in \mathbf{T_x M}, \mathbf{x} \in \mathbf{M}$. Smooth means $\forall\ \mathbf{F} \in \mathbf{C}^{\infty}(M)$, the composite map

$$\mathbf{(vF)(x) = v(x)F : M \text{ -> } R}$$

is $\mathbf{C}^{\infty}$.

The utility of all this mathematical machinery in vision and visual processing comes from the concept of exponentiation. Beginning with the ideas of an integral curve as a parametized curve $\mathbf{x=\phi(t)}$ whose tangent vector coincides with **v**, $\mathbf{d\phi/dt = v(\phi(t))}$, the uniqueness of ODE insures that for t sufficiently small $\forall \mathbf{x}$ there exist integral curves $\mathbf{\psi(t,x)}$, with $\mathbf{\psi(0,x)=x}$. ψ is called the flow generated by **v**.

Note that ψ is a one-parameter group of transformations since

$$\mathbf{\psi(x, \psi(t,x)) = \psi(t+s,x).}$$

The flow is usually written suggestively

$$\mathbf{\psi(t,x) = \exp(tv)x}$$

The Lie bracket or commutator of two vector fields v,w

$$[\mathbf{v},\mathbf{w}](F) = \mathbf{v}(\mathbf{w}(F))-\mathbf{w}(\mathbf{v}(F))$$

Determines group properties via Lie's theorems, in particular

Let **v, w** be vector fields on **M**. Then

$$\exp(t\mathbf{v}) \exp(s\mathbf{w})x = \exp(s\mathbf{w}) \exp(t\mathbf{v})x$$

iff **[v,w] = 0**.

v is said to be an infinitesimal generator of the group **G**.

The transformations needed for imaging, and in particular for vari-pixel imaging, can be viewed as a vector field. Concentrating on only the vertical (or horizontal) lines in Figure 4, the vector field which defines a unique direction to each point in the image manifold is evident. By Lie's theorem, this visual warping, or any other, can be generated locally according to the Lie algebra.

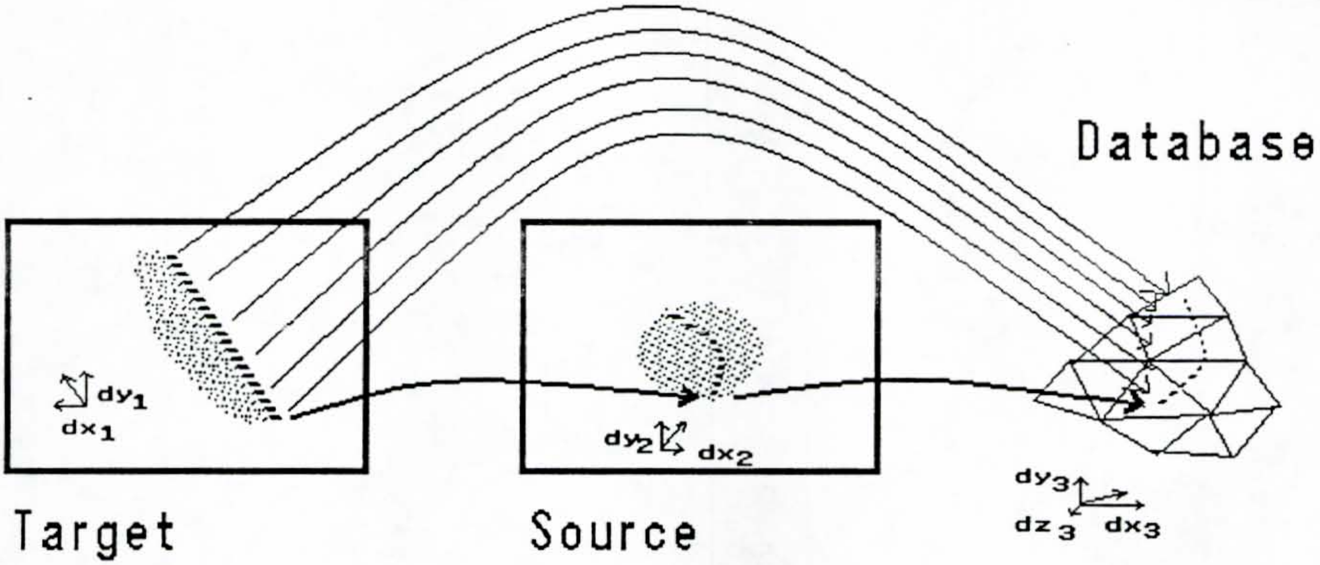


Figure 5. The mappings involved in generation of images with variable pixel sizes. The source image can be bypassed by using Lie group theory.

This ability to generate the warping through local action has important benefits for the construction of algorithms using paralleled processors. Since the Lie algebra determines the vector field locally, the warping need not be solved in a closed form expression. Instead, the warping transform can be generated locally by starting at a pixel in the target image that corresponds with a known pixel in the source image. For example, such points may lie along an axis.

As Figure 5 shows, a line or curve of pixels in the target image induces a trajectory or flow in the source image. The pixels along the source trajectory are the values needed in the target image. The important point is that these flows are locally determined as the differential notation in Figure 5 suggests; global solutions are not needed.

The possibility of using local solutions saves computational resources. Whereas global solutions typically involve expressions of transcendental functions with high computational cost, the local of Lie algebra expressions are relatively simple, taking the form of small matrix and vector operations. As a result, it will be computationally more efficient to utilize the local Lie algebra expressions.

When the image is algorithmically generated (rather than acquired as a video image) more economies are possible. In this case the pixel can be traced through to its source in the data base. The pixel will typically original on a polygonal facet or other object within the data base. The local flow parameters can be carried along to the data base object where they generate a trajectory in three dimensional space as suggested in Figure 5. The source image plane which is intermediate between the data base and the target image need only be visited once for each data base object.

Passing directly from the data base to the target image results in substantial computational savings. Using this direct mapping, generating the pre-distorted display required by vari-pixel imaging will require only a little more computation than generating a standard undistorted image. Pletincks (1988) presents similar techniques that use quaternions rather than vector language.

A major portion of this research will be directed at developing efficient algorithms so that the distortion overhead can be held to 50% or less. Lie's theorems guarantee that direct mapping from database to target is feasible; what remains is to develop efficient algorithms.

An Example of Image Warping

Software was developed to distort images off-line using Lie group techniques. A variant of the software was used to simulate the distortion properties of the optics. This software was implemented on a NeXT workstation using Absoft objective- FORTRAN 77. FORTRAN was used to emphasize the computational aspects of the problem. Absoft's implementation provides interface to the object-oriented environment of the NeXT.

The images distorted were in the form of Postscript graphics files. Postscript is the native graphics language of the NeXT computer and provided an easily readable data format for these demonstration programs.

Figure 6 shows a 320 by 200 pixel image of a WWII P38 fighter as read from a disk of canned images. The postscript image was reproduced on an Apple Laserwriter connected to the NeXT. This reproduction process provides about 4 bits of gray scale through dithering.

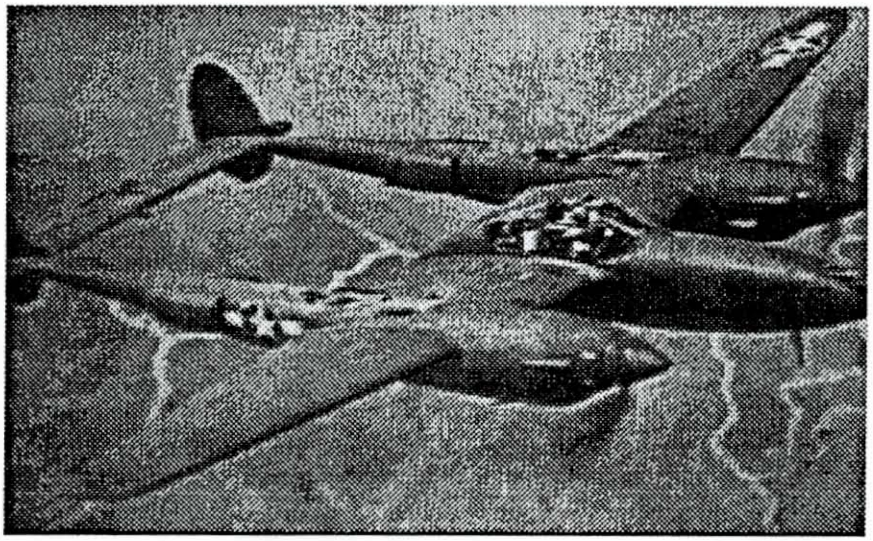


Figure 6. Postscript image of a WWII P38 fighter. Image is 320 by 200 pixels with 4 bit gray scale.

The image in Figure 6 was pre-distorted according to the equation

$$\rho = \mathbf{r_o r/(r_o\text{-}r)}$$

where **r** is the radial coordinate in the source image and ρ is the radial coordinate in the target image. The radius $\mathbf{r_o}$ is a parameter chosen to produce the desired distortion. The polar angular coordinate remains unchanged.

This equation produces a dramatic warping since as $\mathbf{r \to r_o}$ since $\rho \to \infty$. The sample optical system above does not produce such a dramatic distortion, but such blow-ups of radial coordinates can easily be produced with optical systems. As refracted rays approach angles of total internal reflections, optical singularities develop. Also, reflected rays go through singularities as rays become tangent to the reflecting surface.

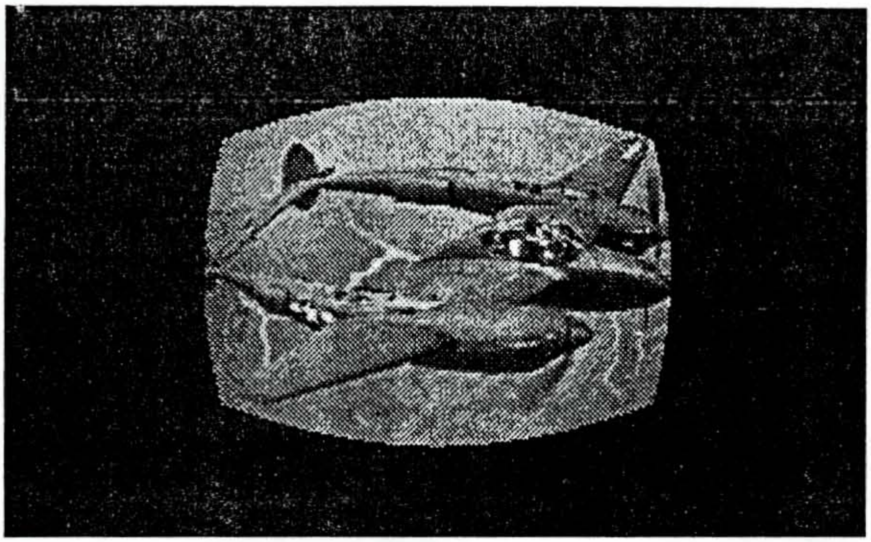


Figure 7. Pre-distorted image of P38.

The pre-distorted image is shown in Figure 7. Note that the center of the image has the same scale and perspective as the original image in Figure 6. The full resolution of the image is thus preserved at the center of the field.

The outer edges of the field are substantially compressed so that the area occupied by the distorted image is less than a third that required by the original image. This variable compression is what permits a wider field of view for a given display.

In a vari-pixel display system, the image of Figure 7 would be viewed through distorting optics something like those in Figure 2. The user would then see a wide field of view image in correct perspective. The center of the image would have high resolution whereas the edge would have lower resolution. This fall off in resolution would match the resolution curve of the human eye shown in Figure 1.

The analog computation the optical system provides cannot be shown here, but a simulation is shown in Figure 8. In this figure, the pre-distorted image has been distorted according to the inverse mapping function:

$$\mathbf{r = r_o \rho/(r_o + \rho)}$$

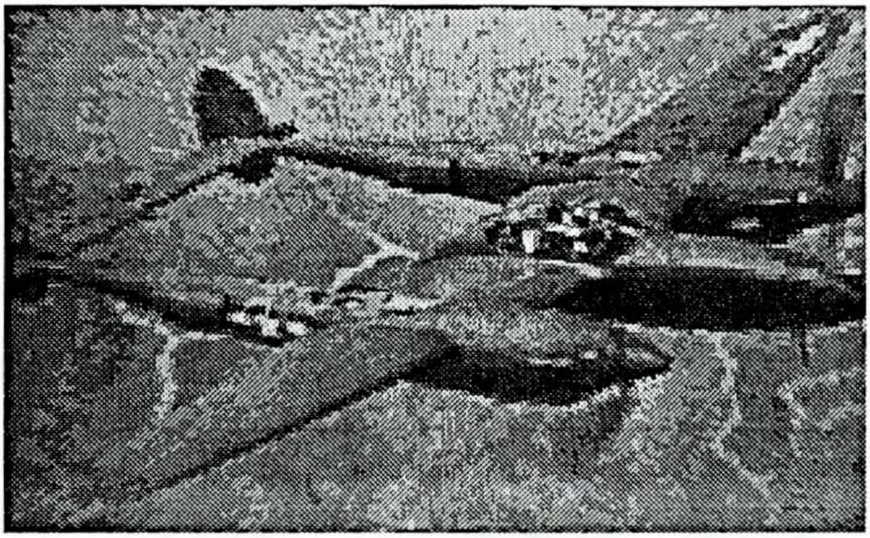


Figure 8. Simulation of view of Figure 7 through distorting optics. Image of P38 has correct perspective but variable resolution.

The image is in Figure 8 has been restored to its original size and perspective, but it looses resolution at the edges relative to the original. This is of course just what is needed to match the human visual system and make optimal use of the available display pixel budget.

Figure 8 contains many aliasing artifacts near the edge of the image. This is due to the digital nature of the simulation; not attempt has been made to anti-alias the image. In the real system, the analog optical calculation would simply magnify pixels falling near the edge of the field. The optics would also provide some inherent anti-aliasing since the optical transfer function would naturally decrease in resolution toward the edge of the field. This would provide high-frequency filtering and anti-aliasing.

Note that all functions used for radial distortion are flat, that is have zero derivative, at $\mathbf{r=\rho=0}$. The distortion function must be smooth or zero-derivative near the axis, since an eye-phone type display does not incorporate eye-tracking. The saccadic movements of the eye will then bring regions near the center of the display to the maximum resolution foveal viewing area. It is important therefore that the resolution be relatively constant near the center so that the resolution taper does not become too obvious as a result of saccadic movements.

Research and Implementation Plans

A system for optimal display will be implemented in stages. During the first year, vari-pixel display technology will be demonstrated using off-the-shelf technology. During the second and third year, more advanced technologies will be used to produce vari-pixel displays with state-of-the-art performance. At the conclusion of the third year, the vari-pixel technology will be ready for marketing.

Early development will be done using easily available inexpensive components. The most readily available display technology suitable for eye-phone type displays is the liquid crystal display (LCD) use in portable, "Watchman"-type televisions. These displays typically have resolutions of about 320 by 240 pixels equivalent to the example images above.

In order to drive a Watchman display, a computer system with a composite video output is necessary. The early low-resolution CGA display had such an output, but is too primitive a technology. Some workstations, such as Silicon Graphics can be programmed to produce a composite display, but such workstations are more advanced than necessaryl for the feasibility demonstration.

A good compromise between excessive sophistication and primitive simplicity is found in the Amiga 3000 computer. The Amiga 3000 uses a 25 MHz Motorola 68030 with 68882 numerical coprocessor. In addition a custom VLSI chip provides graphic BitBLT (Bit BLock Transfer) operations independent of the CPU. The Amiga is easily networked over ethernet with other the other workstations at IST. THe UCF Film Department has had good success in using the Amiga in film animation applications.

The early optics will be of a bread-board nature and will use ready made components mounted in hand-made fixtures. Ready made optics are available from a variety of sources in many materials and configurations.

These components will provide ample support for testing the theory and algorithms developed during the first year of the project. During later phases of the project, higher performance hardware will be obtained and developed so that a state of the art vari-focal eye-phone display can be developed.

Workstation-level graphics performance is the goal for project as a whole. The recently announced NeXTdimension board for the NeXT workstation is a likley choice for the display generation side of the system. The NeXTdimension uses an Intel i860 RISC graphics coprocessor. The i860 is capable of 33 MIPS, 66 MFLOPS and has built-in z-buffer support. NeXT rates the i860 at 30,000 Gauraud polygons/second. These are real polygons, since the screen clear time is 30 MS; all too often polygons/second figures are given for unrealistically small polygons. Like the Amiga, the NeXTdimension supports a variety of display frequencies and formats.

Dr. Thomas Clarke, principal investigator, has had much experience with the NeXT and is developing a RISC-based coprocessor for the NeXT using the Inmos transputer under FHTIC funding. The NeXTdimension thus fits naturally with on-going research efforts at IST. The Inmos transputers may prove a way to boost performance beyond that obtainable with the INtel i860 alone.

Proprietary software will be avoided as much as possible. Graphics standards will be utilized whenever practical so that the vari-pixel system will be as portable as possible. The development language will be C for maximum portability in any case.

The display system will be chosen after a careful evaluation of the state-of-the-art. In order of preference, possibilities available in the near future include: high resolution miniature LCD, miniature CRT, and CRT combined with fiber optic.

It would be ideal if the television industry would produce a high-resolution LCD for HDTV (High Definition TV) use within the time frame of this project. While this will ultimately happen, progress in fabrication technology may not be fast enough to meet the goals of this project.

Small high-resolution CRTs developed for use as helmet mounted displays (HMDs) could be combined in triads with appropriate filters to produce a high-resolution color display. Drawbacks to the use of HMD CRTs are cost and fabrication. These CRTs are special purpose items and tend to be fairly expensive. Also the use of multiple CRTs, filters and combining optics can lead to a complicated structure with difficulties in maintaining optical alignment.

A final option would be to use a large CRT with a fiber-optic relay to the final head-mounted eye-phone display optics. This approach takes advantage of mass produced display technology but has the draw back of a large and somewhat fragile fiber-optic umbilicus. A medical endoscope fiber bundle may be adequate for this application, however. The Visual Technology Research (VTRS facility of the Naval Training Systems Center (NTSC) here in Orlando has experience with fiber-optic relays for head-mounted projection displays, and their expertise can be utilized (Browder, 1989).

At the time of writing the fiber-optic relays seems the most practical. It would be viewed as stop-gap solution since the miniature LCD is much more desirable. A compromise would be to design with the fiber-optic/large-CRT behaving as a virtual miniature LCD.

The optics for this second phase of the project will be specially designed and fabricated by a US manufacturer specializing in this type of optical fabrication. The mounting hardware will be designed for durability and long-life.

The algorithms developed during the first year will be adapted to the high-performace hardware platform. This high performace vari-pixel system will then be made available to industry for marketing and production.

Comparison with Other Research

The emphasis in this project is to use a group theoretic approach to vision as the basis for design of an optimized virtual-world display. Group theory is the natural language since the human visual system is optimized for detecting curves and lines which are invariants of Lie transformation groups identified by Hoffman (1966).

Several approaches to implementing Lie groups are possible as discussed above. The projective geometry transformations used in visual systems are examples of a type of group transformation, and Lie group transformations can be implemented in a similar fashions by expressing the components of the transform in matrix format. Lie groups are inherently local, however, so that if parallel hardware is available, there will be no problem in using it to rapidly process different portions of the image.

A more recent approach to implementing vision transforms relies on the mathematics of conformal transforms (Frederick and Schwartz, 1990). These researchers utilize the Riemann mapping theorem to implement the transforms found in the visual system. The approach of Frederick and Schwartz is also very amenable to implementation on parallel hardware.

Standard approaches using projective geometry techniques can be found in Dew et. al. (1989), Glassner (1989) and Magnenat-Thalmann (1987). Full use will be made of appropriate existing work in the field of computer image generation and visualization.

Summary

In summary, the first year will be devoted to developing algorithms for implementing image pre-distortion based on Lie group theory. These algorithms will be tested using general purpose computer hardware and low cost displays.

During the second and third years, a state of the art system will be developed utilizing the algorithms and configuration proven during the first year.

I. PREVIOUS ACCOMPLISHMENTS and QUALIFICATIONS

Related Research

The papers "Stereo Processing with Artificial Neural Networks" (1990 Florida Artificial Intelligence Research Symposium), and the paper "Generalization of Neural Networks to the Complex Plane" (1990 International Joint Conference on Neural Networks in San Diego), both apply" reverse engineering" of human signal processing to the solution of recognition problems.

Dr. Clarke has also done work on the recognition of underwater acoustic echoes as in "A Pattern Recognition Approach to Acoustic Bottom Recognition"; 1987 in Acoustical Imaging, H. W. Jones editor, Plenum, NY. In addition, Dr. Clarke has applied quantum mechanical techniques such as path integration to underwater acoustic problems in Wave Propagation in Focusing Random Media; 1982; NOAA Tech Memo ERL AOML-51.

Optical design work by Dr. Clarke is represented by his 1983 paper describing a new eyepiece design he developed.

Facilities available to Dr. Clarke include NeXT computers, DSP hardware, and a variety of workstations used in support of mathematical simulation. They are described more fully below.

Significant Papers

Clarke, T.L., (1990a). Stereo Processing with Artificial Neural Networks. *Proceedings 1990 Florida Artificial Intelligence Research Symposium.*

Clarke, T.L. (1990b). Generalization of Neural Networks to the Complex Plane. Proceedings 1990 Int. Joint Conference on Neural Networks, San Diego.

Clarke, T. L. (1987). A pattern recognition approach to remote acoustic bottom characterization, in Progress in Underwater Acoustics, Harold M. Merklinger ed, (Plenum, New York), 225- 230.

Clarke, T. L., (1983) Simple flat field eyepiece, *Appl Optics, 22,* 1807-1811.

Clarke, T. L. (1982). Wave propagation in focusing random media, NOAA Technical Memorandum, AOML-51.

Biographical Sketch

Thomas L. Clarke

SUMMARY: Dr. Clarke has more than 15 years research experience involving propagation modeling, digital processing of acoustic signals, statistical analysis, and numerical modeling. He has two patents. Dr. Clarke has published extensively in professional journals and is affiliated with the Institute of Electrical and Electronics Engineers, Audio Engineering Society, and Acoustical Society of America.

Education

Ph.D. in Applied Mathematics, May 1982, University of Miami.

M.S. in Applied Mathematics, August 1975, University of Virginia.

B.S. in Mathematics, Dec. 1973, Florida International University.

Experience

1988- present. Senior Scientist, University of Central Florida/Institute for Simulation & Training. Dr. Clarke's activities involve planning and developing new research projects. Areas of research with UCF faculty include the application of advanced computer architectures to problems of training simulator design and application of applied mathematics to simulation.

1985 - 1988. Consultant and contract researcher on various acoustical projects. Obtained SBIR grant to study optical propagation in the atmosphere doing business as TLA Research.

1975 - 1987. Mathematical Oceanographer at Ocean Acoustics Division of Atlantic Oceanographic and Meteorological Laboratory, U.S. Dept. of Commerce. Engaged in research relating to Doppler current sensing and echo-sounding techniques.

Publications

Dr. Clarke has published over 50 articles and conference presentations. The following are among his most scientifically significant. These are primarily from his stint as an underwater acoustical applied mathematician but still reflect his broad range and versatility as a researcher.

1977 Clarke, T. L., FET Pair and op-amp linearize voltage- controlled resistor, Electronics, 50, No. 9, 111-113.

1978 Clarke, T. L., Oblique factor analysis solution for the analysis of mixtures, Mathematical Geology, 10,225-241

1981 Clarke, T. L., Parallel channel processing overlaps data acquisition and reduction, Computer Design, 20, No. 8, 127- 132.

1981 Clarke, T. L., Augmented passive-radiator loudspeaker systems, Part I, J. Audio Eng. Soc., 29, 394-404.

1981 Clarke, T. L., Augmented passive-radiator loudspeaker systems, Part II, J. Audio Eng. Soc., 29, 511-516.

1982 Clarke, T. L., Wave propagation in focusing random media, NOAA Technical Memorandum, AOML-51.

1982 Clarke, T. L., D. J. P. Swift, R. A. Young, A numerical model of fine sediment transport on the continental shelf, Environ. Geol., 4, 117-129.

1982 Clarke, T. L., G. L. Freeland, B. Lesht, D. J. P. Swift, and R. A. Young, Sediment resuspension by surface wave action: An examination of possible mechanisms, Mar. Geol., 49, 43- 59.

1983 Clarke, T. L., W. L. Stubblefield, and D. J. P. Swift, Use of power spectra to estimate characteristics of sand ridges on continental shelves, J. Geology, 91, 93-97.

1983 Clarke, T. L., Simple flat field eyepiece, Appl Optics, 22, 1807-1811.

1983 Clarke, T. L., D. J. P. Swift, and R. A. Young, A numerical modeling approach to the fine sediment budget of the New York Bight, J. Geophys. Res., 88, 9653-9660.

1984 Clarke, T. L., and D. J. P. Swift, The formation of mud patches by non-linear diffusion, Cont. Shelf Res., 3, 1-7.

1984 Clarke, T. L., Limitations of physical theory, Nature, 308, 1984 Clarke, T. L., J. R. Proni and J. F. Craynock, A simple model for the acoustic cross-section of sand grains, J. Acoust. Soc. Am., 76, 1580-1582.

1984 Clarke, T. L., An application of measure theory to the problem of flexible working hours, J. Irreproducible Results, 29, No. 2, 15.

1986 Clarke, T. L., Radiation pressure induced instability in Saturn's rings, Astrophysical Ltrs., 25, 51-56.

1986 Clarke, T. L., and J. R. Proni, Remote acoustical measurement of ocean bottom parameters, in Current Practices and New Technology in Ocean Engineering, T. McGuinnes and H. Shih eds, (ASME, New York) 145-148.

1986 Clarke, T. L., and G. J. Williams, Transverse Doppler current profiler phase I development final report, General Oceanics Technical Report.

1987 Clarke, T. L., A pattern recognition approach to remote acoustic bottom characterization, in Progress in Underwater Acoustics, Harold M. Merklinger ed, (Plenum, New York), 225- 230.

1987 Clarke, T. L., Final report phase I multi-wavelength refraction correction, TLA Research Report.

1989 Clarke, T.L., GRASS Research at IST, Proceedings 5th Annual GRASS User's Conference.

1989 Clarke, T.L., The Application of RISC Processors to Training Simulators, FHTIC Final Report.

1989 Clarke, T.L., Terrain Data Bases for Simulation, FHTIC Final Report.

1990 Clarke, T. L., Stereo Processing with Artificial Neural Networks, Proceedings 1990 Florida Artifical Intelligence Research Symposium (FLAIRS).

1990 Clarke, T. L., Generalization of Neural Networks to the Complex Plane, Proceedings 1990 International Joint Conference on Neural Networks (IJCNN).

Patents

Dr. Clarke in addition holds two patents. This illustrates his practical turn of mind.

US Patent 4070697. Augmented Passive Radiator Loudspeaker. Feb 28, 1976

US Patent 4623224. Anastigmatic Eyepiece. Nov 18,1986.

Proposals Pending

The principal investigator is currently receiving funding from the Florida High Technology and Industry Council (FHTIC) for research into RISC processors and Terrain Data Bases. He currently devotes about 60% of his time to these two projects. Renewal proposals for these projects are pending.

The PI has a proposal pending with the NSF for development of an undergraduate simulation-based matematics and science course.

IST Facilities

The Institute for Simulation and Training (IST) is part of the University of Central Florida (UCF). IST employs students from a variety of advanced degree programs in engineering, science and mathematics. Graduate students from mathematics and computer science will play a significant role in this research project.

IST has extensive laboratory facilities that can be divided into four broad categories: Visual Systems Laboratory, Networking Laboratory, Human Factors Laboratory, and the Mathematical SImulation Laboratory.

Visual Systems Laboratory

The Visual Systems Laboratory contains a variety of computer systems used in support of research into visual system technology. These systems include:

- Silicon Graphics 4D70GT and two Personal IRISs. RISC based UNIX workstations with specialized graphics hardware used in support of simulator data base development and prototyping and experimentation of image generation algorithms.
- Sun 386i. Intel 80386 based UNIX workstation used for applications dealing with geographical information processing.
- NeXT. Motorola 68030 based UNIX workstation used for research into object oriented user interfaces or "glass cockpits".
- MacIntosh IIx. Motorola 68030 computing appliance used for experiments in object-oriented simulation.

- Various PC/AT-class personal computers used for hardware interface and control.

Networking Laboratory

The Networking Laboratory contains a variety of US Army furnished SIMNET compatible equipment including:

- Two M1 SIMNET tank simulators together with MCC network controller. Also scheduled for delivery are: SIMNET Stealth Vehicle, SIMNET Data Logger, SIMNET Plan View Display, SIMNET Long Haul Gateway, and Perceptronics ASAT (Aviation Situational Awareness Trainer). In addition, the Networking Laboratory possesses a variety of general purpose computing resources including:
- DEC VaxStation 3100. VAX VMS or ULTRIX workstation used in support of network data analysis.
- Seven Hewlett-Packard Vectra Intel 80386 XENIX or MS/DOS PCs. These are used for program development and testing and for network protocol conversion.
- Harris NightHawk 68030 computer.
- Evans and Sutherland ESIG-500 image generator.

Human Factors

Under the Human Factors umbrella fall a variety of equipment used in on-going experiments in training effectiveness. This equipment includes two TOP GUN part task gunnery trainers, two General Aviation Trainers (GATS), two VIGS tabletop part task gunnery trainers, two Electronic Information Delivery Systems (EIDS), and a network of Accer computers used for team training research.

Also falling in this category is the Human Performance Laboratory which contains an SAIC Delta II neural network coprocessor hosted by a 386 PC.

Mathematical Simulation

The Computer Systems category includes a variety of equipment that is particularly relevant to this project. This equipment includes:

- Three 80386 PC compatible computers. A Hewlett-Packard Vectra QS-16 and two special purpose home-built computers. These machines are used to host a variety of hardware add-in cards such as a PC-Vision Plus video digitizer, an OKI Semiconductor digital signal processing (DSP) card, and an Inmos transputer coprocessor.
- Two NeXT Motorola 68030 UNIX workstation. In addition to its object oriented features, the NeXT contains a Motorola 56001 DSP chip and provides access to the workstation network in the Visual Systems Laboratory. These are being upgraded to 68040 mainboards and plans are afoot to convert the old 68030 mainboards into home-brew NeXT compatibles.

J. BIBLIOGRAPHY

Boff, Kenneth R., Lloyd Kaufman, and James P. Thomas, 1986. *Handbook of Perception and Human Performance, Vol I*: John Wiley and Sons, New York, p 7-49.

Browder, Blair, 1989. Evaluation of a helmet-mounted laser projector display,in *Helmet-Mounted Displays*, Jerome T. Carollo, editor. SPIE, Bellingham, Wash, 14-18.

Clarke, T. L., 1983. Simple flat field eyepiece, *Appl Optics, 22*, 1807-1811.

Clarke, T.L., 1990a. Stereo Processing with Artificial Neural Networks. *Proceedings 1990 Florida Artificial Intelligence Research Symposium*.

Clarke, T.L. 1990b. Generalization of Neural Networks to the Complex Plane. *Proceedings 1990 Int. Joint Conference on Neural Networks*, San Diego.

Dew, P. M., R. A. Earnshaw, and T. R. Heywood (editors), 1989; *Parallel Processing for Computer Vision and Display*; Addison-Wesley Publishing Company, New York, 503pp.

Dodwell, Peter C., 1983. The Lie transformation group model of visual perception, *Perception and Psychophysics, 34*, 1-16.

Frederick, Carl and Schwartz, Eric. L. 1990. "Conformal Image Warping", *IEEE Computer Graphics and Applications*, March , 54-61.

Glassner, Andrew S. (editor), 1989; *An Introduction to Ray Tracing*; Academic Press, New York, 327pp.

Hoffman, W. C. ,1966. The Lie algebra of visual perception. *J. Math. Psychol., 3*, 65-98.

Hoffman, W. C. ,1989. The visual cortex is a contact bundle. *Applied Math. Comput.,32*, 137-167.

Hubel, D. H. and Wiesel, T. N. ,1959. Receptive fields of single neurons in the cat's striate cortex. *J. Physiol. (London), 148*, 574-591.

Lenz, Reiner ,1990. Group invariant pattern recognition. *Pattern Recognition, 23*, 199-217.

Magnenat-Thalmann, N. and D. Thalmann; 1987; Image Synthesis; Springer-Verlag, New York, 400pp.

Olver, Peter J. ,1989. *Applications of Lie Groups to Differential Equations*. Springer-Verlag, New York, 497 pp.

Plenticks, D., 1988. The use of quaternions for animation, modelling and rendering, in New Trends in Computer Graphics, N. Magnenat-Thalmann and D. Thalmann, editors, Springer-Verlag, New York, 44-53.

Rotier, Donald J., 1989. Optical Approaches to the Helmet Mounted Display, in *Helmet-Mounted Displays*, Jerome T. Carollo, editor. SPIE, Bellingham, Wash, 14-18.

VanName, Mark L., and Bill Catchings, 1990. "Virtual Reality Reprsents New Level of Communication", *PC Week, 7*, 41, p53.

Welford, W.T., 1974. *Aberrations of the Symmetrical Optical System*, Academic, New York.

Copies of the articles by Dodwell (1983), Frederick and Schwartz (1990), Rotier(1989), Pletinicks (1988) and Browder (1989) are attached. Each of these articles provided important background to this proposal.

Dodwell presents a very readable account of the utility of Lie groups in vision based on Hoffman's work.

Frederick and Schwartz establish the utility of conformal mappings for image generation.

Rotier presents a survey of the many varieties of optical systems that are adaptable to head-mounted computer displays.

Pletinicks discusses how quaternions, another formulation of differential geometry, can be used for image transformations.

Browder discusses the applications of fiber optics to head mounted displays in the context of the on-going research at the Naval Training Systems Center VTRS.