DESIGN AND IMPLEMENTATION OF KIMS MULTICOMPUTER SYSTEM

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

BHASKAR CHOWDHURI

to the

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

SEPTEMBER, 1992

POST
I.I.T Kanpur
Submitted on 10/9/

[illegible]

Certificate

It is certified that the work contained in the thesis entitled "*Design and Implementation of KIMS Multicomputer System*", by Sri Bhaskar Chowdhuri, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

प्रौद्योगिकी संस्थान
INSTITUTE OF TECHNOLOGY

[illegible] [illegible]

Dr. R. N. Biswas	Dr. Rajat Moona
Department of Electrical Engg.	Department of Comp. Sc. and Engg.
I.I.T. Kanpur	I.I.T. Kanpur

Acknowledgement

I express my sincerest gratitude to my guides, Dr. Rajat Moona and Dr. R. N. Biswas, for their invaluable guidance and constant encouragement, without which this work could not have been possible.

I take this opportunity to specially thank Mr. S. Kumar, Mrs. S. Sule Mr. R. Singh of A.C.E.S. and all the staff and faculty members of computer science department for their kind help and support. Special thanks are due to Mr.R. N. Tiwari and Mr. Shiv Shankar , who took personal care in the development of KIMS.

I also thank Mr. Deepak Gupta, who, with his profound knowledge of UNIX and LATEX, helped me in more ways than I can remember.

I thank my Hall-IV friends– Rahul, Rajan, Ashim, Gurudev (Sekhar'da) and all the other jolly good chaps of Hall-IV, who shared my joys and sorrows during the entire work and always provided a brotherly support. Their love and support have made my stay at I.I.T. Kanpur a memorable experience. I thank them all from the bottom of my heart.

Part of the work was financially supported by Motorola (India) Inc., which is duly acknowledged.

Finailly, I would like to dedicate this thesis to my father, Dr. Narayan Chaudhuri, my idol in life.

I.I.T. Kanpur BHASKAR CHOWDHURI

September 1992

09 DEC 1992
CENTRAL LIBRARY
KANPUR
Acc. No. A.114547

Th
001.64404
C459d

CSE-1992-M-CHO-DES

Abstract

Low-cost parallel computer systems, built using easily available resources are of great demand. Towards this goal, we have designed and implemented KIMS multicomputer system. The system consists of a interconnected cluster of IBM PC/ATs. We designed a special communication card, which augments the ATs with communication features. The communication card supports number of data transfer modes which includes high speed DMA transfer mode. These communication modes are software selectable. The link interface also supports multicasting in hardware. To enable easy use of the system, we have provided a high level language support in C. The system is currently running at the computer science department of IIT kanpur.

Contents

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

List of Figures

Chapter 1

Introduction

In this chapter we discuss the motivation behind the present work. We also provide a brief overview of the work and finally discuss the organisation of the thesis.

1.1 Motivation for the Present Work

To satisfy the demands of enhanced computing power, various multicomputer and multiprocessor architectures have evolved in the near past. Multiprocessors and multicomputers fall under the Multiple Instruction Multiple Data stream (MIMD) category of parallel computers. Both consist of a network of Computing Elements (CE) which cooperate and communicate amongst themselves to solve a problem. However, the CEs in a multicomputer system are complete with local memory and processing power. Therefore, they can also work as stand-alone computers, whereas, CEs in a multiprocessor system do not have any independent existence.

The CEs in multicomputer and multiprocessor systems communicate either through a shared memory or by passing messages through communication links. Although, the shared memory systems are more attractive from the software-developers viewpoint, message-passing systems are much easier to implement and show better performance with large number of computing nodes.

Many multiprocessor and multicomputer systems are commercially available, however, their prohibitive cost restricts their wide use. Recently, a number of research efforts have been directed towards low cost implementation of multicomputer and multiprocessor systems.

Towards this goal, we have designed and implemented a multicomputer system using IBM PC/ATs. The reasons behind choosing IBM PC/AT's as the computing elements are

- Low price and wide availability.
- Availability of literature providing internal hardware and software design issues.
- Easy access to the system bus and its detailed description.
- The high computing power of PC/ATs with advanced processors like 80386, 80486.

1.2 Overview of the Work

As discussed in the previous section, the objective of the present work is to design and implement a message-passing multicomputer system using IBM PC/ATs. On the hardware side, the work involves the implementation of high performance FIFO based communication links with features particularly suitable for multiprocessing application. On the software front, we have developed a language support to enable easy use of the system. In particular the system has the following features.

1. Each computing node can have upto 16 bidirectional communication links. So, networks with rich connectivity can be realised.
2. Each link has hardwired FIFO buffer which significantly reduces the synchronization delays.

3. Multicasting is directly supported in hardware.

4. The links can be programmed to provide various data transfer modes, namely, polling, interrupt driven and DMA controlled transfer modes. Thus, one can choose the best mode for a particular application.

5. Links have very high data transfer rate ($>$ 1.0 MByte/sec in DMA controlled mode).

6. There is no particular architecture or network topology dictated by the system. So it can be used to realise a variety of network topologies.

7. A high level language interface in C has been provided through which one can use the system without knowing the underlying hardware details.

The system is presently running in its minimum configuration at the computer science department of IIT, Kanpur. Initial performance of the system is very encouraging. The system acts as a real test-bed for parallel software development and provides significant computing power with the existing resources.

1.3 Organization of the Thesis

The organization of the remaining chapters is as follows.

In Chapter 2, we present a survey of the related works. We also provide a critical comparison of these works with the present work. We introduce the system architecture in chapter 3 and provide the actual hardware implementation details in chapter 4. In chapter 5, we discuss the software environment provided in the system. Specifically, we discuss the C language support provided to implement configuration transparent programs. The performance measurements of the existing machine configuration are quite encouraging. We discuss this in Chapter 6. Finally, we conclude by suggesting modifications for a better design in chapter 7. We also provide some of the design details in Appendix A and the PCB design in Appendix B.

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Chapter 2

Survey of Previous Works

In this chapter, we survey some of the existing microprocessor based multicomputer systems similar to our system. After presenting a brief design philosophy behind each of these systems, we analyze their performance and compare them with our present system.

2.1 Microprocessor Based Multicomputers

With the advent of microprocessors, low cost implementation of multicomputer systems became feasible. In recent years, many such systems have been reported in the literature. Amongst these, the Cosmic Cube architecture proposed by Seitz [8] is inexpensive and easy to implement. It also gives good speedup for many applications. The system consists of 64 small computers connected in the form of a 6-dimensional hypercube. An operating system kernel in each node schedules and runs processes within that node and provides system calls for processes to send and receive messages. However, the communication links have large communication latency as, they are slow serial links without multicasting features.

It is widely accepted that a high throughput can be achieved by overlapping computation with communication. Absence of such a technique is also felt in the

Cosmic Cube machine. This was introduced in the Acorn multicomputer system proposed by Goodeve and Taylor [3]. They have used separate coprocessors to speed up the links. But many other problems remain unaddressed in the system. The absence of buffers to decouple the sending and receiving processes, software overhead in maintaining shared data and absence of multicasting make the links unsuitable for large multicomputer systems.

A similar approach is also used in the hypercube multiprocessor system by Das et al. [1]. The system consists of 8 CEs connected in 3-dimensional hypercube topology. But instead of serial links, it uses twoport memories shared between two adjacent CEs for data communication purpose. This system behaves marginally better than the previous system as it was observed that the flat structure of twoport memory is not particularly attractive for point-to-point communication links. In fact large software overhead is required to maintain the shared data structures. Moreover large number of interface signals (address, data and control lines) makes it unattractive for building large systems. Absence of multicasting is another shortcoming of the system.

A simple low-cost multicomputer system based on message passing FIFO links was proposed by Ghosal et al. [2]. The system consists of eight IBM PC/XT mother boards connected together using point-to-point byte-wide bidirectional communication links with FIFO buffers. A language interface in Pascal with a few routines for interprocess communication and remote process initialisation is also provided with the system. The communication links of the system cannot be satisfactorily used with large systems because of the following shortcomings.

- Absence of multicasting.
- Number of links is only one per I/O channel.
- Only blocked mode of data transfer is provided.

improvements. The unified view provided for all the links with common control, mask and status registers makes the physical distribution of the links transparent. The mask register reduces the input-output space requirement drastically. The user is provided the option of selecting different input output modes of the links which includes a high speed DMA transfer mode. This is done by properly changing the control register value. Both blocked and unblocked modes of transfers are provided.

The software environment provided for the system is quite similar to that of the MMS system, but the software overhead is much smaller than that in the MMS system as many of the features which were emulated through software in MMS are directly supported in the hardware.

2.3 Conclusion

From the survey of some of the recent microprocessor-based multicomputer systems, it appears that the present system offers many interesting features needed for building large systems which were not supported in the earlier systems.

Chapter 3

KIMS Architecture

We introduce the architecture of *Kanpur IIT Multicomputer System* (KIMS) in this chapter. We start our description by presenting a broad overview of KIMS. We then discuss the details of its communication structure. Specifically, we discuss details of different registers associated with the communication interface. We also show how they are used to get different input output modes and multicasting feature. Finally, we discuss the special considerations that are to be made for implementing KIMS with IBM PC or XTs.

3.1 Architectural Overview of KIMS

KIMS uses IBM PC/ATs as the computing elements. Configuration of ATs used in our present implementation is 16 MHz 80386SX processor with 4 MB main memory. We have augmented the ATs with communication links to use them in KIMS. The links are realised in specially designed extension cards where each card hosts 4 bidirectional byte-wide links. Each AT can have upto 4 such cards to provide a maximum of 16 links. The cards are identical in design. Only two switch settings distinguish one card from another. Each link is provided with 2KB FIFO buffer to decouple the communicating processes.

The link interface does not use any special features of a PC/AT and can be used with any AT or its clone. The link interface can also be used with a normal 8 bit PC at the cost of reduction in the number of links and interrupt driven communication facility. However, polling and DMA controlled communications are not restricted. We discuss this in more details at the end of this chapter.

The communication interface does not dictate any particular network topology, so a number of multicomputer configurations can be realised with KIMS. We have tested the system with two computing nodes.

3.2 Communication Link Registers

As described in the previous section, the communication links are supported in specially designed extension cards. These cards are connected to the AT's mother board through standard extension slots. For easy use of the communication links, several user-accessible registers are provided. These registers are physically distributed across different cards, but the novelty in implementation makes this totally transparent to the user.

The 6 registers provided in the link interface are listed below.

1. Reset Register (pseudo output).
2. Control Register (8 bit output).
3. Mask Register (16 bit output).
4. Fifo-empty Register (16 bit input).
5. Fifo-full Register (16 bit input).
6. Link Register (8 bit input / output).

All these registers are mapped in a block of 8 byte locations of the processor's address map. The base address of this is programmable. We have chosen it as 140 Hex in present implementation. However, in our discussion to follow, we denote it as BASE.

3.2.1 Reset Register

The *reset register* is a pseudo output register. It occupies the address (BASE + 4). Writing in this register will result in the reset of all communication links. The same effect is also obtained when the processor is switched on or the reset button of the processor is pressed. As it is a pseudo register, no data gets physically stored. The reset register can be used as 8 bit as well as 16 bit register.

3.2.2 Control Register

The *control register* is an 8 bit output register occupying address (BASE + 2). Its main function is to select different communication modes. We use only the lower 4 bits of this register and the upper 4 bits are considered as "don't care" bits as shown in figure 3.1. We describe the function of each of these bits below.

WARN. This bit is used to enable 'illegal i/o WARNing' (WARN) interrupt. If this bit is set, an interrupt will occur for 'illegal link-read' or 'illegal link-write' operation. By 'illegal link-write', we mean an attempt to write in a link whose FIFO buffer is already full. Similarly, by 'illegal link-read', we mean an attempt to read from multiple links at the same time or attempt to read from a link with empty FIFO. Physically, this interrupt is mapped with AT's IRQ 10 interrupt line. WARN bit should be kept low if IRQ10 is used by any other device.

DAV. This bit is used to enable 'Data has ArriVed' (DAV) interrupt. If this bit is set, an interrupt is generated whenever any selected (see description of mask

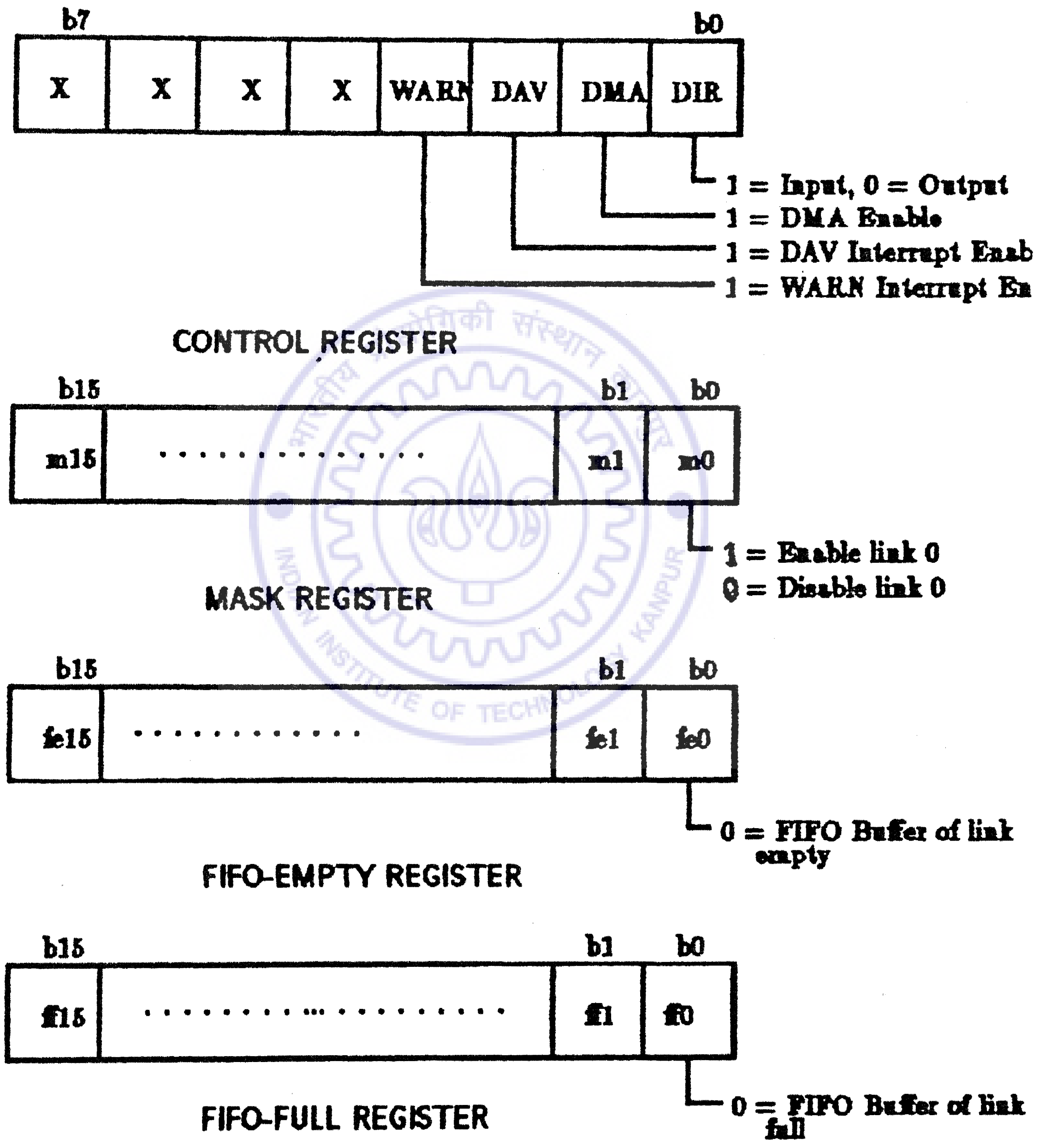


Figure 3.1: User accessible registers in the link interface

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

register) empty link gets data. The DAV interrupt is very helpful, because it eliminates the overheads of polling the link status for data arrival. Physically, this interrupt is mapped with AT's IRQ11 interrupt. This bit should be kept low, if IRQ11 is used by any other device.

DMA. This bit, when set to one, selects DMA mode of data transfer. In DMA mode the DIR bit is used to decide the direction of DMA transfer. Setting DIR to one indicates link read operation whereas, reseting DIR to zero indicates link write operation. Once these two bits are properly set, DMA request signal will be generated as long as a valid transfer is possible. Valid transfer is indicated by the non-empty selected FIFO for read operation and non-full selected FIFOs for write operation. Deactivation of DMA request occurs when the above conditions are violated, while automatic reactivation of the DMA request occurs as soon as the condition again becomes favorable for further transfer. Selecting more than one link for link read operation causes no transfer. Physically, the DMA request is mapped to DMA channel 3 on PC/AT bus. If it is used by some other device, then line conflict can be avoided by keeping the DMA bit low.

DIR. This bit is used in conjunction with DMA bit to denote DMA direction as explained above. If DMA bit is zero then the value of DIR bit is considered as 'don't care'.

3.2.3 Mask Register

Mask register is a 16 bit output register occupying address location BASE. This register is used to enable or disable the links for read or write operation. Bit m i of mask register corresponds to link i. If m i is set to 1 then link i is selected for access otherwise it is masked. More than one link can be enabled for writing. But it is illegal to enable more than one link for read operation. Under this condition, no data links. These modes are Polling mode, Interrupt driven mode and DMA controlled link access modes.

3.3.1 Polling

In *Polling* mode, Fifo-empty and Fifo-full registers can be checked by the driver software, before each byte of transfer. Data transfer should be done only if the buffer condition is favorable for the transfer. This mode is simple to use and performs well with small data transfers.

3.3.2 Interrupt Driven Access Mode

In *interrupt driven access mode* links are accessed without their status checking. Occurance of WARN interrupt indicates that the last link operation failed because of any of the reasons discussed above. This mode is much faster than Polling due to the elimination of the overhead of status checking for each byte of transfer.

3.3.3 Direct Memory Access Mode

In *direct memory access mode* data transfer takes place without CPU intervention and is the fastest mode supported in hardware. Link read operation through DMA continues as long as the required number of bytes are not read or the selected link's FIFO becomes empty. DMA operation is resumed again when further data becomes available in the link buffer. Similarly, for link write operation, DMA process temporarily halts when any of the selected link's FIFO buffer becomes full. DMA operation permanently stops when required number of bytes get transferred. This mode gives more than 1 MB/sec data transfer with 16MHz 80386SX PC/ATs. However, a reprogramming of DMA controller is needed for each session of transfer. This involves the loading of word count register, address register, mode register etc. in the DMA controller. For small data transfer (< 10 bytes), this overhead outweighs the gains of DMA mode data transfer.

3.4 Multicasting

The term *multicasting* means simultaneous transfer of data through multiple links. It also implies the capability to select any subset of the links, through which multicasting will take place. It is apparent from the above discussion that multicasting is very easy in KIMS. Multicasting is done by first enabling the desired links by setting their corresponding mask bits in the mask register. Any write in the link register now causes data to be simultaneously written in all these selected links. Automatic synchronisation is done to make sure that all the selected links are coping up with the transfer. Whenever any of the selected link's FIFO gets filled up, data transfer is temporarily halted until at least one location becomes available in that FIFO. This way single instruction multicasting is supported in KIMS.

3.5 Special Considerations for PCs and XTs

Originally the communication interface was designed to work in conjunction with IBM PC/ATs. But similarity between 8 bit PC's interface with that of ATs, makes the interface work with 8 bit PCs as well. However, with each PC, we can attach only 2 such cards to have upto 8 links instead of 16 links as in the case of ATs. This restriction occurs because the card assumes 16 bit data bus and folds upper and lower 8 bits internally to support the distributed registers (this is discussed in detail in the next chapter). Moreover the WARN and DAV interrupts are not directly supported in PCs as they do not have IRQ10 and IRQ11 interrupt lines. However, one can map these interrupts to some other unused interrupts to have interrupt driven transfer facility. Polling and DMA driven modes are directly supported and no modification is needed here.

A system built in this way, using IBM PCs, will obviously not offer the computation speed achieved with ATs, but still the system can be useful as a low-cost test bed for parallel software development.

3.6 Conclusion

In this chapter, we presented architecture of link interface in KIMS. The actual implementation details are given in chapter 4. We have also discussed about different communication modes provided in KIMS. The communication interface can also be used with IBM PCs or XTs at the cost of loosing some communication features and performance.

Chapter 4

Hardware Implementation of Communication Interface

In this chapter we discuss the implementation details of KIMS communication interface. Specifically, we discuss the details of bus-interface, distributed link registers, DMA control and interrupt generation logic.

4.1 Design Philosophy

We have designed a PC/AT extension card for the communication interface. This card is used to augment the PC/ATs with communication links. The design goals for the card are described below.

- The cards should be identical in design for ease of management and fabrication.
- A card should incorporate as many links as possible.
- The cards should be robust. The hardware should be able to ignore illegal software commands.
- The design should be reliable. Strict bus interface logic should be used to reduce possibility of errors.

The communication interface is implemented on a PC/AT extension board, whose size, specified by IBM, is 13.12×4.8 inches. On a double sided printed circuit board with plated through holes, we could accommodate a maximum of 4 links.

The entire control logic for this interface card is implemented in PALs. We used PALs instead of discrete logic because PALs provide greater flexibility in design and consume less space and power. Although Gate Arrays would have provided a better optimization and integration, their high cost is prohibitive in applications like ours.

4.2 Organization of the Card

For ease of visualization, we classify the user accessible registers into 2 groups. The first group registers are *reset, control, mask, fifo-empty* and *fifo-full* registers and are collectively called general registers. The other group contains only the *link register*. We present the broad organisation of the communication board in figure 4.2. There are 5 basic functional blocks in the interface, which we list below.

1. Address Decoding Block.
2. Data Bus Interface.
3. General Registers.
4. Link Interface.
5. Interrupts and DMA Control Block.

In the following sections we describe each of these 5 blocks separately with occasional reference to other blocks for non-local signals.

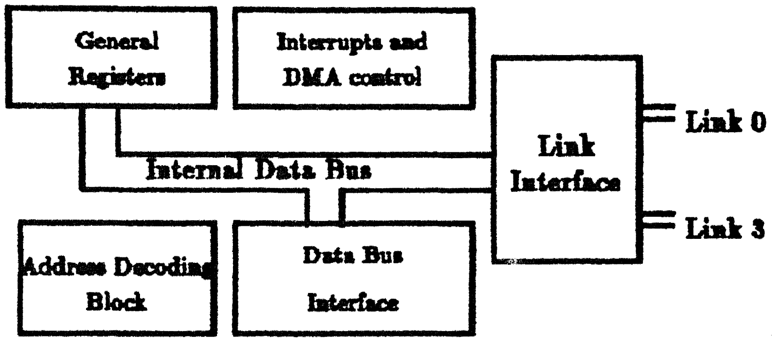


Figure 4.1: Block diagram of the communication card

4.3 Address Decoding Logic

The *address decoding block* consists of two PALs, namely **EPLD1** and **EPLD3**. Part of **EPLD3** is also used by the *general register block*. The address decoding block generates two signals, namely $\overline{\texttt{gen}}$ and $\overline{\texttt{link}}$, both of which are active low signals. Signal $\overline{\texttt{gen}}$ indicates selection of one of the general registers. Therefore, it becomes active when the system's address bus contains a valid address lying in the range BASE to (BASE+5). Signal $\overline{\texttt{link}}$, on the other hand, indicates selection of the link register. It is, therefore, activated when address bus contains a valid address equal to (BASE+6) corresponding to the *link register address*. Since the links are also mapped to the processor's DMA channel 3, signal $\overline{\texttt{link}}$ is also activated with $\overline{\texttt{DACK3}}$ signal. We provide the details of **EPLD1** and **EPLD3** logic in Appendix A.

4.4 Data Bus Interface

Communication interface in KIMS allows the use of upto 4 communication cards with each CE. These cards are numbered 0 to 3 and are selected through switches on the cards. *Data Bus Interface* block provides the interface between these cards and the processor data bus of a CE. Interface logic switches two halves of the processor's

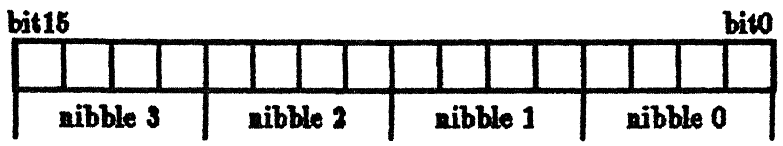


Figure 4.2: Nibbles of 16 bit register data

data bus to a cards' internal bus depending upon the communication register being accessed and the card's number. We discuss this below.

- For *link register* and *control register* access, only the lower 8 bits of the 16 bit processor's data bus are used. This is true for all the 4 cards.
- For *mask, fifo-empty* and *fifo-full* register access (all 16 bit registers), nibble i of the processor's data bus is used by card i as shown in figure 4.4.
- For *link register* write operation, the internal data bus is mapped to the lower byte of the processor's data bus on all the 4 cards. The *link register* read operation is a little more complex. At a time only one link can be read. As the links are distributed on different cards, first it is ensured that only one link is enabled out of 16 links distributed across 4 cards. Only in that case, internal data bus is mapped to the lower byte of the processor's data bus on the card with enabled link. In all other cards, internal bus is kept isolated from the processor's data bus.

In order to meet the above mentioned requirements, we use separate drivers with tristate outputs for each nibble of the processor's data bus. The resulting bus interface is shown in figure 4.4. Transceiver DBFi is used to interface nibble i to the processor data bus. The internal data bus of each interface card is 8 bit wide. We represent this by cd0..cd7. Mapping of the 16 bit processor bus to 8 bit internal bus is explained later. The link interface also uses another 4 bit bus, denoted by mx0..mx3, which is derived from the 8 bit internal data bus. This 4 bit bus is used exclusively by the card's *mask register*.

There are 2 control signals associated with each transceiver, namely $\overline{\mathtt{dir}}$ and $\overline{\mathtt{sg}}$. The signal $\overline{\mathtt{dir}_i}$ decides the direction of data flow in buffer DBF*i*. For all the 4 buffers, we drive these lines with the processor's $\overline{\mathrm{IORD}}$ signal. This ensures that data from the communication cards appears on the processor's data bus only during an input operation. Signal $\overline{\mathtt{sg}_i}$ corresponds to the output enable signal of buffer DBF*i*. The logic to generate $\overline{\mathtt{sg}}$ signal for four buffers is implemented in EPLD2. We now discuss the mapping of the processor bus for individual registers.

The *control* register is an 8 bit write-only register duplicated on all cards, which is mapped at an even address. During a write in the *control* register, data appears on the lower 8 bits of the processor's data bus. Therefore, lower 8 bits of the processor's data bus are mapped into the internal 8 bit bus during a control register write operation. This mapping is achieved by simultaneously activating $\overline{\mathtt{sg}_0}$ and $\overline{\mathtt{sg}_1}$ on all the cards.

For *link* register write operation, similar mapping is done on all the cards. However, for *link* read operation this mapping is done only on the card with selected link. Further, the mapping is done only after ensuring that there is only one enabled link in the CE. This condition is indicated by $\overline{\mathtt{tmpillrd}}$ signal generated by EPLD5 discussed in the section on *DMA and interrupt control block*.

For *fifo-empty* and *fifo-full* register read operations, a card provides only one nibble of data. This is achieved by activating $\overline{\mathtt{sg}_i}$ on card *i*. However, internal connection in the board ensures that all the 4 buffers (DBF0 to DBF3) get same nibble value. Thus, by activating $\overline{\mathtt{sg}_i}$ on card *i*, complete 16 bit register content is placed on processor data bus.

For *mask* register write operation, the desired nibble from the 16 bit processor data bus is extracted in two steps. For card number 0 and 1, this nibble is available

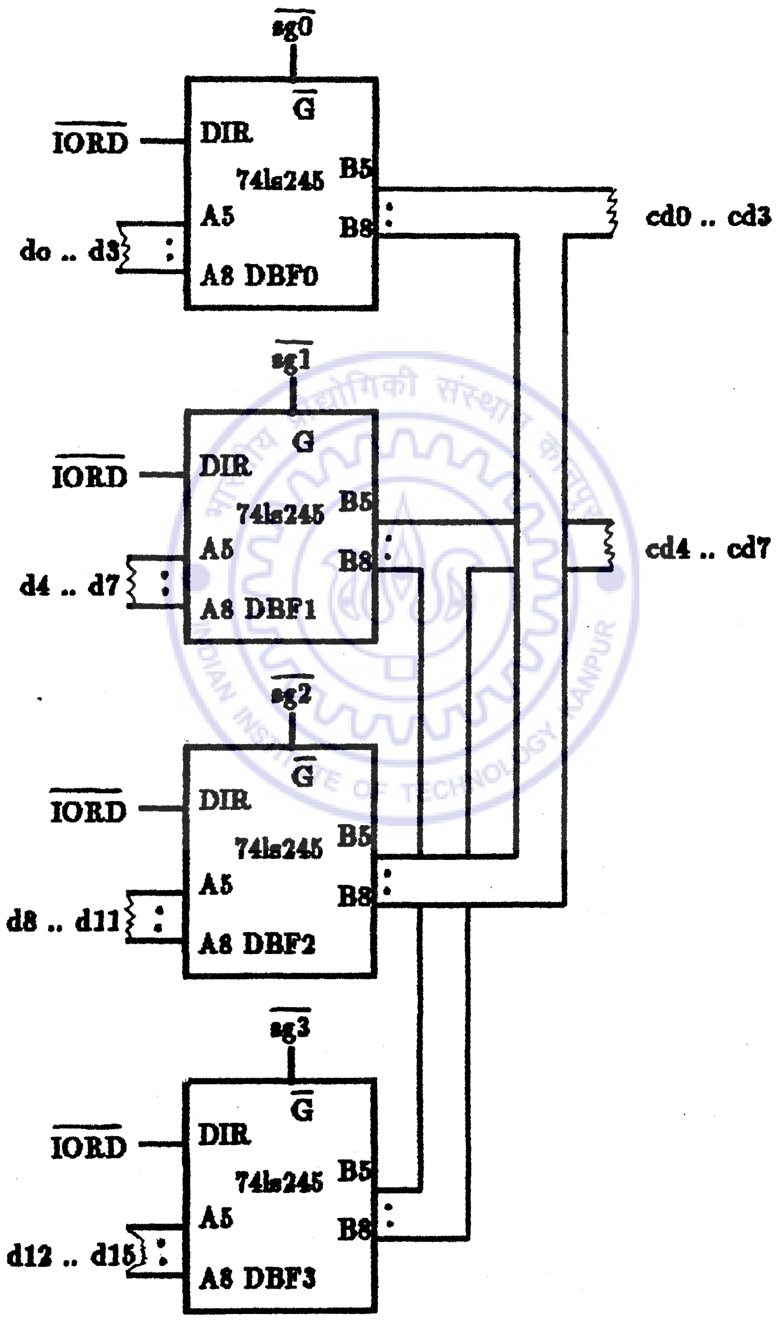


Figure 4.3: Eight bit bus derivation

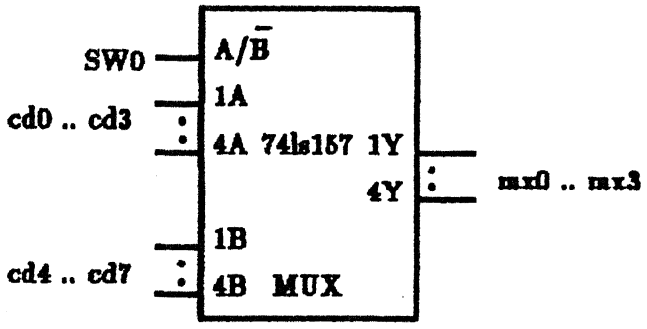


Figure 4.4: Four bit bus derivation

in the lower byte of the processor data bus. However, for card number 2 and 3, this lies on the upper byte of the processor bus. Therefore, in the first step we extract the byte containing the desired nibble and map it to the internal data bus cd0..cd7 on each card. In the second step, we extract the proper nibble from cd0..cd7 and map it to the 4 bit bus mx0..mx3 using a 74ls157 (MUX). MUX uses bit 0 of card address to select the desired nibble (see figure 4.4).

4.5 General Registers

Once the data bus interface is properly designed, it is relatively simple to implement the *general registers*. All input registers are realised using 74ls245 buffers, while the output registers are realised with 74ls374 latches. We use PAL EPLD3 to generate the read/write signals for the general registers. We discuss the implementation details of each of the general registers below.

4.5.1 Reset Register

The Reset register is a pseudo output register and does not require any storage. A write into this register causes EPLD3 to activate the $\overline{\texttt{reset}}$ signal, used for reseting the FIFOs (IDT7203). As the reset register is duplicated on all 4 cards, a write into

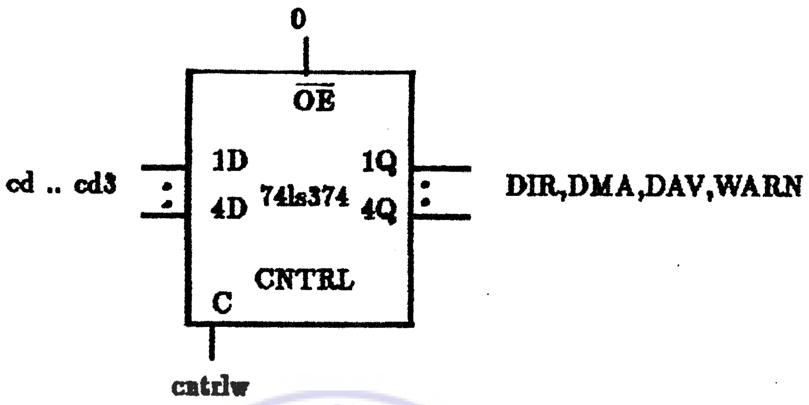


Figure 4.5: Implementation of control register

it leads to a global reset of all communication links. Following a reset, FIFO buffers go into "empty FIFO" condition.

4.5.2 Control Register

The Control Register is an 8 bit output register referred to as **CNTRL**. The `cntrlw` signal generated by **EPLD3** is used as the clock input for **CNTRL**.

4.5.3 Mask Register

The Mask register is a 16 bit distributed output register. A 74ls374 latch, referred to as **MASK**, is duplicated on all 4 cards and holds nibble i of mask register's content on card i. The signal `maskw`, generated by **EPLD3**, is used as the clock input for **MASK**. When a write in this register is detected, this signal is made active and the data present on the internal 4 bit data bus (`mx0..mx3`) gets latched in **MASK**.

4.5.4 Fifo-empty and Fifo-full Registers

These two registers are 16 bit input registers which provide the 'FIFO empty' and 'FIFO full' conditions of the FIFOs associated with each link. As each card has 4

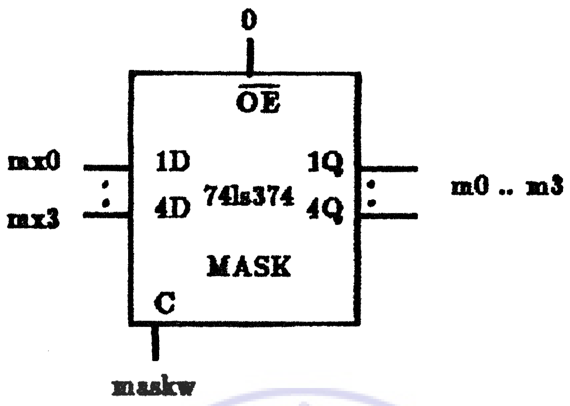


Figure 4.6: Implementation of mask register

links, it can provide only 4 bit information of these registers. This 4 bit information on card i forms the nibble i of 16 bit data. We provide the implementation details of these two registers in figure 4.5.4 and 4.5.4. In both the registers, 4 status signals from 4 links on a card are duplicated on two nibbles of the internal 8 bit bus. This ensures that all the data bus buffers get same value. However, only one nibble appears on the output bus as discussed earlier in *data bus interface*. EPLD3 generates signal $\overline{\texttt{stat1rd}}$ to enable *fifo-empty* register and $\overline{\texttt{stat2rd}}$ to enable *fifo-full* register.

4.6 Link Interface

We refer to the physical connection and the associated interface which connects one CE to another, as *link*. A link has two hardwired FIFO buffers, each holding data in one direction. Whenever a CE requires to send some data to another CE connected through a link, it writes the data in the FIFO buffer present in its link interface. The other CE, referred as the remote CE, knows presence of this data from the 'FIFO empty' status sent to it. On finding that the FIFO contains some data, it can read this data by sending 'read request'.

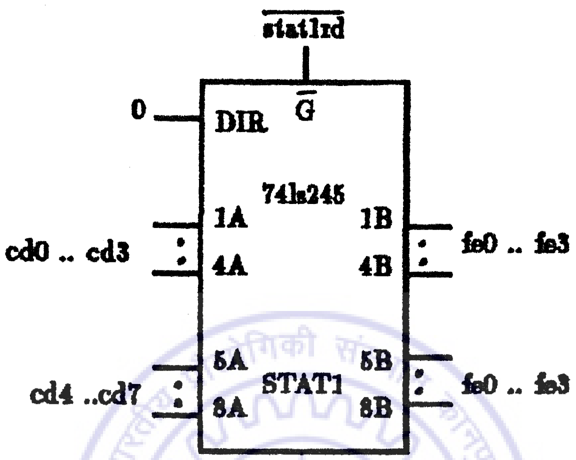


Figure 4.7: Implementation of fifo-empty register

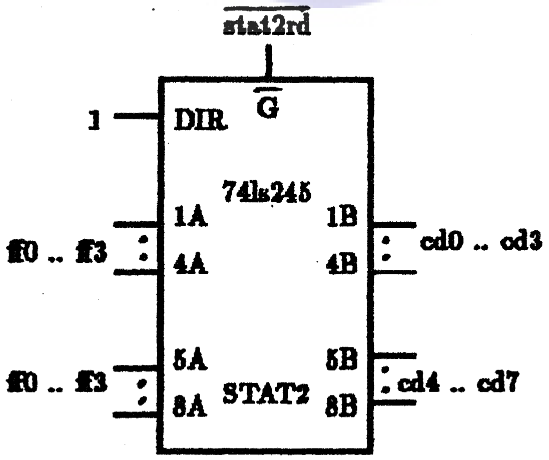


Figure 4.8: Implementation of fifo-full register

The physical connection of a link is brought out on a 40 pin edge connector. Every alternate pin of this connector is grounded to shield against external noise perturbations. Thus, there are 20 active signals, of which 10 are incoming and 10 are outgoing. These signals are, 8 bit data lines, 1 read signal and 1 'FIFO empty' signal in each direction.

We show the block diagram of one of the four identical links interface in figure 4.6. We use **OTBUF** to drive the outgoing data lines of the local FIFO and **INBUF** to receive the incoming data lines. The transmission and reception of the 'read request' and 'FIFO empty' signals are done using **INOTBUF**. We denote the read and write signals for link i as $\overline{\texttt{linkrd}_i}$ and $\overline{\texttt{linkwr}_i}$ respectively. These signals are generated by **EPLD4**. Whenever a write is done in the *link register*, a write signal is generated corresponding to each enabled link on the card. Similarly for link register read operation, a read signal is generated corresponding to the enabled link on the card. However, this is done only after ensuring that there is exactly one enabled link in the processor. Validity of this condition is indicated by $\overline{\texttt{tmpillrd}}$ signal generated by **EPLD5** in *DMA and interrupt control block* described below.

4.7 DMA and Interrupt Control Block

The *Interrupt and DMA control block* consists of two sub-blocks. (1) *inter-card communication interface* and (2) logic for interrupt and DMA generation.

As discussed earlier, it is often required to ensure that only one bit is enabled in the mask register (or only one link is enabled). For this, information is needed from other cards, as a card holds only 4 bits of the 16 bit *mask* register. There are 4 signals used for this purpose as described below.

- The $\overline{\texttt{PREV}}$ signal is an input signal to card i. $\overline{\texttt{PREV}}$=0 indicates that one or more link on card 0 to card $i-1$ has been selected. For card 0 this input is tied to

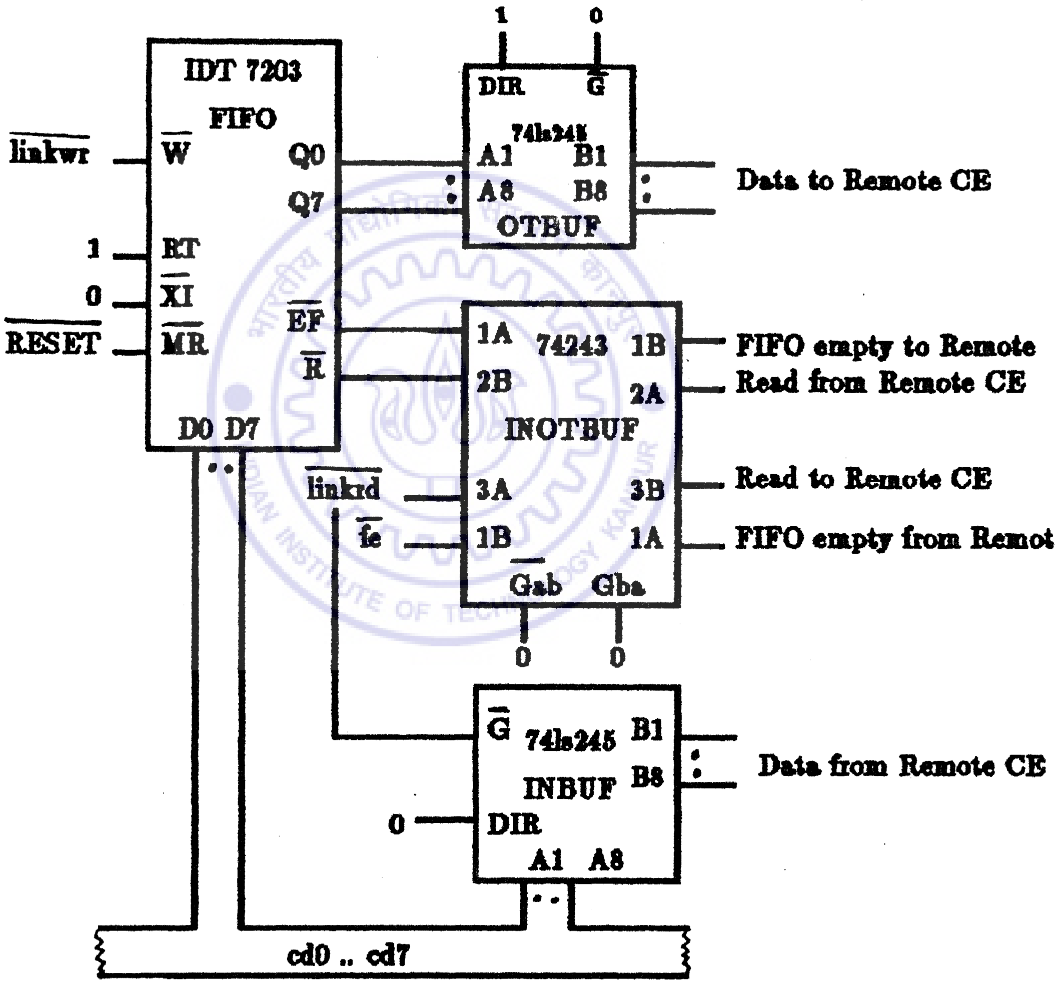


Figure 4.9: Link interface

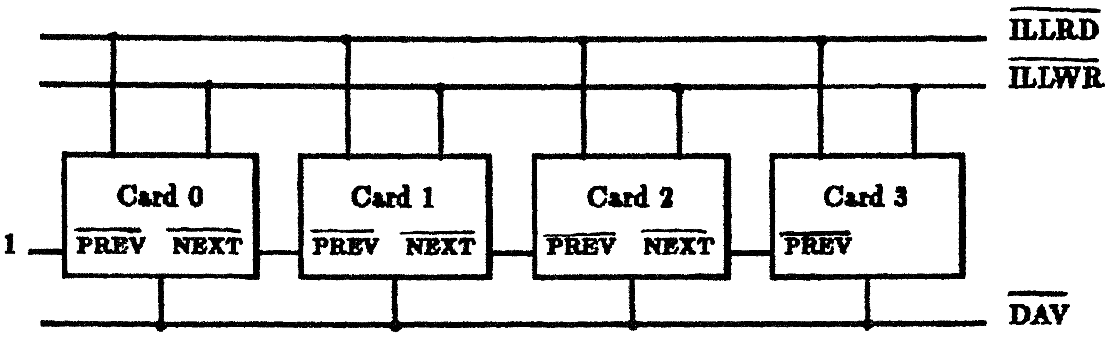
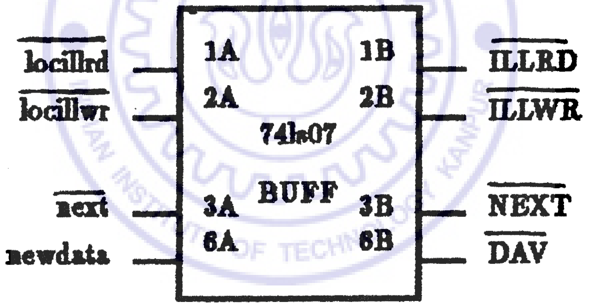

$\overline{\text{ILLRD}}$
$\overline{\text{ILLWR}}$
Card 0
$\overline{\text{PREV}}$ $\overline{\text{NEXT}}$
Card 1
$\overline{\text{PREV}}$ $\overline{\text{NEXT}}$
Card 2
$\overline{\text{PREV}}$ $\overline{\text{NEXT}}$
Card 3
$\overline{\text{PREV}}$
1
$\overline{\text{DAV}}$

Figure 4.10: Inter-card communication signals



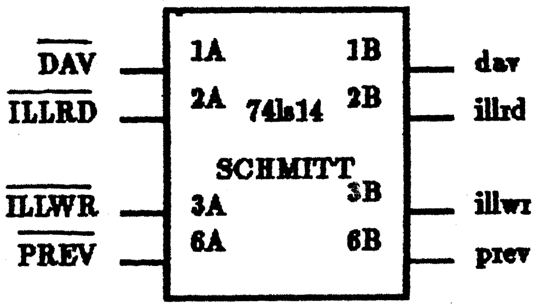


Figure 4.11: Bus interface of inter-card communication block

one.

- The $\overline{\text{NEXT}}$ is an output signal of card i. $\overline{\text{NEXT}}$=0 indicates that one or more link on card 0 to card i has been selected by the *mask* register. This signal is tied to $\overline{\text{PREV}}$ of card $i+1$. The signal is derived from $\overline{\text{PREV}}$ and the mask bits on the card i as

  $\overline{\text{NEXT}}=\overline{\text{PREV}} \vee m_i \vee m_{i+1} \vee m_{i+2} \vee m_{i+3}$.

- $\overline{\text{ILLRD}}$ is a shared line. A zero on this line indicates an 'illegal condition for link read operation'. By 'illegal read condition' we mean at least one of the two following conditions. (1) Either more than one link has been selected for reading or (2) the selected link does not have any data for reading. This line is driven by open-collector output on each card. When a card detects an illegal condition for link read operation, it pulls this line low, otherwise this line is floated. Thus, this signal is implemented using wire-ORing of local illegal read signals. Since all the cards monitor this line, an 'illegal read condition' on any card becomes known to all other cards.

- The $\overline{\text{ILLWR}}$ signal is similar to $\overline{\text{ILLRD}}$, except that it denotes 'illegal link write condition'. By 'illegal link write' condition we mean that at least one of the selected links has its FIFO buffer full and no further write is immediately possible. Signal $\overline{\text{ILLWR}}$ is implemented by wire-ORing of local illegal write signals.

- The $\overline{\text{DAV}}$ is a shared line. A zero on this line indicates that at least one of the selected links has some data, which can be read. This line is driven by open-collector output on each card. When a card detects data arrival in one of its selected link, it pulls this line low, otherwise this line is floated. Since all the cards monitor this line, so a data arrival on one card becomes globally known.

The DMA request and interrupt signals are generated by the logic realised with **EPLD5** and **EPLD6**. First, we discuss the function of **EPLD5**. Among the signals generated by this PAL are `maskor` and $\overline{\mathtt{newdata}}$ signals. On card *i*, signal `maskor` is derived by the logical ORing of bits in nibble *i* of *mask* register. Signal $\overline{\mathtt{newdata}}$ denotes the arrival of data in one of the selected links of the card. Another signal, $\overline{\mathtt{tmplillrd}}$ denotes that more than one link has been selected. This signal is derived from $\overline{\mathtt{PREV}}$ signal and the card's *mask* value. Two other signals, $\overline{\mathtt{loclillrd}}$ and $\overline{\mathtt{loclillwr}}$, denotes that card's link condition is unsuitable for read and write operation respectively. These signals are also used in generating $\overline{\mathtt{ILLRD}}$ and $\overline{\mathtt{ILLWR}}$ as discussed earlier.

EPLD6 serves two purposes. It generates $\overline{\mathtt{NEXT}}$ signal for the *inter-card communication interface* on all the cards. In addition to this, on card 0 it is also used to generate DMA request, WARN and DAV interrupt signals.

The logic details of **EPLD5** and **EPLD6** are given in the Appendix A.

4.8 Conclusion

In this chapter we discussed the communication interface of KIMS. We used six 16L8 PALs to realise the entire control logic of the communication interface. A very robust bus structure is used to decrease the possibility of error. Sufficient precautions have been taken to guard against any possible damage to the hardware due to software errors.

Chapter 5

Software Environment

In this chapter we discuss the software environment in KIMS. The KIMS software environment consists of a sequential programming language augmented with routines for remote process creation, termination and inter-process communication. Specifically we discuss the language support in C, which is currently available on KIMS. Finally, we present a small example program to elaborate the usage of this environment.

5.1 Introduction

The current understanding of the software issues for loosely coupled computer system is not very advanced. Our effort in designing the software environment has been to make the environment as general purpose as possible. This is certainly desirable for solving different classes of problems on KIMS. However, many software issues depend upon the algorithm used as well as on the underlying hardware. So a truely general purpose software environment is not only difficult to implement but can become overwhelmingly inefficient in many situations. Therefore, our effort has been to provide a language support which balances these two conflicting issues. We discuss the resulting environment, which is quite similar to the software environment of the MMS system proposed by Moona [6].

5.2 Need

There has been considerable research in computer science on new concurrent languages. One basic barrier in parallelizing the conventional sequential declarative languages has been the use and modification of global variables. Maintaining the coherency of the global variables takes large overhead. It is therefore necessary to consider languages that have no concept of global variables and where the processing and variables are localized to the modules and can be executed concurrently. This has motivated *object oriented* and *functional* languages. However, this style of programming has been found comparatively inefficient for scientific computations.

A newer concept called *dataflow* is evolving. Here, a computation is executed when it gets all the operands needed for the operation. Our approach in building a software environment can be viewed as a *large grain dataflow environment*, where a processes is synchronized using messages from other processes. Thus in some sense, we can visualize the messages in KIMS as tokens in dataflow computers.

The software environment discussed in this chapter views computing elements as nodes in which *one and only one* processes can be executed at a time. It also assumes homogeneous CEs in the system. By homogeneous CEs we mean that the processor used and memory size is same for all CEs. This restriction results, because the memory image of the program in the CE0 is directly transferred to other CEs during a remote fork operation.

The software environment of KIMS is composed of several layers. An user can build the upper layers using the features provided by the lower layers. As system architects, we provide the lowest layer with basic routines for process management and inter-process communication. This layer does not provide automatic process mapping or parallelism detection. They can be design issues for the higher layers. However, even with the current language support, one can use KIMS with little knowledge about the underlying hardware implementation details.

5.3 Basic Functions

As described in the previous section, the lowest layer of the software environment in KIMS provides basic routines for process creation, termination and message passing. In this section we consider these routines as supported for the C language. However, it is obvious that a similar language support can also be implemented for other programming languages.

5.3.1 Process Creation and Termination

RFORK subroutine (Remote Process Creation Subroutine)

```
int rfork(mask)
    unsigned int mask;
```

This subroutine creates a copy of the code, data and stack segments of the executing process on the remote CEs which are connected through enabled links. The **mask** value encodes the links which are to be selected for this function.

Newly created remote processes continue execution from the same program location from which the original process executes after return. All data initialisation done by the calling process are also reflected in the remote process. The **rfork()** subroutine returns 1 to the calling process and 0 to the newly created remote nodes. This return value is important in deciding appropriate decomposition of the process.

A Complementary routine to terminate a process is called by a process when its execution is complete or some fatal error is detected.

TERMINATE subroutine (Remote Process Termination Routine)

```
void terminate(mask);
    unsigned int mask;
```

This subroutine when called by a process, causes the CE to terminate the running process. The CE then starts monitoring 'FIFO empty' condition of the enabled links for further data arrival. The value of **mask** encodes which links are to be enabled. The data from any of these enabled link is interpreted as a memory image for another process. In the current implementation, we used polling technique to check for any data arrival, but one can also use DAV interrupt for the same purpose. This routine is executed by all the CEs at the power on time except the CE with *nodeid=0*. This particular CE is the master node from which the system is operated. A program initiates from CE 0 and then migrates to other CEs for concurrent execution using a `rfork()` call.

5.3.2 Interprocess Communication

The processes running concurrently on different CEs need to communicate between themselves for data and partial results. We provide a wide variety of communication routines for this purpose. All the communication routines are implemented by the complementary pair of *send message* and *receive message* instruction. We discuss these routines below. In all the routines described below, parameter **var** denotes the starting address of the data in the memory, from where data transfer has to start. Parameter **size** specifies number of bytes to be transferred. Parameter **mask** encodes the links through which this transmission has to take place. This value is loaded into the *mask* register at the beginning of a routine.

Unblocked Communication Routines

UBREAD subroutine (Unblocked Read Subroutine)

```
 unsigned int ubread(mask,var,size)
       unsigned int mask,size;
       char *var;
```

This routine is used for reading data var from the channel specified by the mask. This is an unblocked read, i. e. if the required amount of data is not available in the enabled link, then it reads only the available data and does not wait for further data to come. The control is returned to the calling program with return value equal to the number of bytes still to be read. A return value equal to 0 indicates successful completion of the function. This routine is helpful in program development stage, as it tries not to cause any indefinite postponement.

UBWRITE subroutine (Unblocked Write Subroutine)

```
unsigned int ubwrite(mask,var,size)
        unsigned int mask,size;
        char *var;
```

This routine is complementary to the ubread() routine and is used for writing data from the memory into the channels selected by the mask value. While this write operation is going on, if the FIFO associated with any enabled link becomes full, then the write operation is terminated and control goes back to the calling program. The routine returns to the calling program a value equal to the number of bytes yet to be transferred. So a return value of 0 indicates successful completion.

Blocked Communication Routines

The unblocked mode of data transfer are implemented using the *interrupt driven data transfer technique,* which is faster than than the *polling* method of transfer. But the main disadvantage of these routines is the overhead associated with resuming the read/write operation on unsuccessful return. We also provide a set of blocked read/write routines to avoid switching overheads. In blocked read/write mode, a function does not terminate till the required number of bytes are transferred. In current implementation, these routines are implemented using *polling* technique. However, one can also use the unblocked read/write routines to implement these routines which will give better transfer speed. We describe the blocked mode read/write routines below.

BREAD subroutine (blocked Read Subroutine)

```
void bread(mask,var,size)
     unsigned int mask,size;
     char *var;
```

This routine is similar to the ubread() subroutine except that it operates in blocked mode. It reads size number of bytes from a channel specified by mask into the array var. It returns to the calling routine only after completion of required number of byte transfer.

BWRITE subroutine (blocked Write Subroutine)

```
void bwrite(mask,var,size)
     unsigned int mask,size;
     char *var;
```

This routine is complementary to the bread() subroutine. It writes size number of bytes from array var into the channels enabled by mask. The routine terminates and returns to the calling routine only after completion of required number of byte transfer.

DMA mode Communication Routines

All the above mentioned routines are supported using POLLING and INTERRUPT DRIVEN I/O technique. We provide two routines which are quite similar to bread() and bwrite() routines. However, this routines are implemented using high speed DMA mode of communication offered by the link interface. These routines are much faster than the other read/write routines. We describe these below.

DREAD subroutinc (DMA mode Read Subroutine)

```
void dread(mask,var,size)
    unsigned int mask,size;
    char *var;
```

This routine is similar to the bread() subroutine except that the data transfer is done in DMA mode. It reads size number of bytes from a channel specified by mask into the the array var. It returns to the calling routine only after completion of required number of byte transfer.

DWRITE subroutine (DMA mode Write Subroutine)

```
void dwrite(mask,var,size)
    unsigned int mask,size;
    char *var;
```

This routine is complementary to the **dread()** subroutine. It writes **size** number of bytes from array **var** into the channels enabled by **mask**. The routine terminates and returns to the calling routine only after completion of required number of byte transfer.

5.4 Example

In this section we present a small example program written using the above mentioned functions. The program computes the sum of first hundred integers. Purpose of this example is solely to give an idea about how programs are written in KIMS. As the execution proceeds, first it creates a remote process using **rfork()** call. The return value distinguishes between the parent process and the forked process. The parent process calculates the partial sum of first 50 integers. The forked proces, on the other hand, calculates the partial sum of 51 to 100 and sends it to the parent process. The parent process receives this value and adds it to its own partial sum giving the final result.

```
#include <KIMS.H>

main() {

   int forkresult,psum,i,othersum,finalsum;

    /* fork the program to the
       CE connected through channel 1 */
   forkresult=rfork(1);
   if (forkresult==1){
      /* this is the parent program */
       psum=o;
       for(i=1;i<=50;i++)
           /* calculate the partial sum of 1 to 50 */
           psum+=i;
       /* read the partial result calculated
          by the forked process */
       bread(1,(char *)&othersum,sizeof(othersum));
       /* calculate the final result */
       finalsum=psum+othersum;
    }
    else {
      /* this is the forked program */
      psum=o;
      for(i=51;i<=100;i++)
      /* calculate the partial
         sum of 51 to 100 */
         psum+=i;
      /* write the partial result to the
         original program*/
      bwrite(1,(char *)&psum,sizeof(psum));

      terminate(1)/* all forked processes should
                     terminate with this function */
    }

}/* end of main */
```

5.5 Conclusion

In this chapter we discussed the software environment for KIMS. The environment consists of several layers. Routines provided in the lower layers run more efficiently than those in the upper layers. However, the upper layers provide more abstract view of the architecture. We also discussed the lowest layer of this environment which has been implemented and supports the basic process creation, termination and communication primitives. Finally, we presented a small example program to demonstrate the use of KIMS software environment.

Chapter 6

Performance Evaluation

In this chapter we present the performance analysis of KIMS. The analysis is done in two stages. In the first stage, we provide the statistics of data-communication speed in various modes. In the second stage, we study the overall performance of the system by solving different standard programs on KIMS and observing the speed enhancement over uniprocessor system.

6.1 Performance Evaluation Strategy

There is no single parameter by which the performance of a multiprocessor system can be denoted. Moreover, the performance of the system also greatly vary depending upon the algorithm used to solve the problem. In absence of any standard benchmarking procedure, we have carried out performance evaluation in two stages. In the first stage, we measured the performance of the system hardware. This includes measurement of raw computation speed and communication bandwidth of the links for different communication modes. In the second stage, we measured the overall system performance by solving a class of standard problems on KIMS. The execution time is then compared with corresponding uniprocessor computation time. The ratio of these two times gives the speedup, which is an indication of the system efficiency including its software environment.

6.2 Basic System Performance

In this section we present the basic hardware performance of KIMS. All these measurements refer to the present configuration of KIMS currently running at computer science department of IIT Kanpur.

Each CE consists of a PC/AT with 4 MB main memory space. The processors used in the CEs are 16 MHz 80386SX. About 3% of the total computation time is used for dynamic memory refreshing. We do not use any separate arithmatic processor with the CEs. However, they can be easily incorporated in the system to enhance the computing power.

The communication links, which are used to send messages between CEs are found to be quite reliable. For measuring the reliability of the communication links, we used two CE setup. One CE in this setup sends a predefined message to other CE in DMA mode of transfer. The message is checked for errors at the second CE for predetermined pattern. No transmission error was detected even for 48 hours of continuous run.

The communication bandwidth of the links depend on the communication mode used. We used the earlier setup for measuring the link speed in three different modes. We first programmed the control register suitably on both the CEs. Then a message of size 1.2MB was sent by one CE and the time taken for this communication was measured using the 8253 timer present on the AT mother board. The clock resolution was 18.2 clock ticks per second. Communication routines were written in assembly language with minimum software overheads. We summarise the result of this experiments below.

In POLLING mode of transfer, communication bandwidth of the links was found to be approximately 60 KBytes per second. This mode is the slowest mode of transfer. INTERRUPT DRIVEN transfer mode is slightly faster than this in which the communication bandwidth of the links was measured to be roughly 110 KBytes per second. The DMA mode of transfer is the fastest mode of transfer supported in KIMS. In this mode, the links exhibit communication bandwidth nearly equal to 1 MBytes per second.

6.3 Overall System Performance

For gaining an insight of the overall performance of KIMS, we solved a class of standard problems on KIMS.

1. Matrix addition.

2. Matrix Multiplication.

3. Sorting of N numbers.

These problems are compute intensive and have a large degree of inherent parallelism. The execution time of the problems on KIMS were compared with corresponding execution time, when solved on a uniprocessor system. In this section we discuss one such experiment. The problem chosen for this purpose is the multiplication of two square matrices. Since the main objective of the experiment is to evaluate the system performance and not to solve the problem in optimum way, therefore, we choose the standard $O(N^3)$ algorithm of matrix multiplication for this purpose, where N is the order of the matrices.

The algorithm is quite simple. Two square matrices, A and B of order N, are multiplied to obtain matrix C. Here the parent process first forks a child process in the remote CE. Then it computes the first N/2 rows of C. The child process on the other hand computes the last N/2 rows of C. Then it sends this partial result to the parent process. The parent process, in turn, reads this partial result and combines it with its own result to get the final value of C. We present the actual program written in KIMS language interface below.

```
int a[N][N],b[N][N],c[N][N];
main()
{
   int i,j,k;
   int fork_result;

    fork_result= rfork(); /* create the child process */
    if (fork_result==1){
      /* parent process */
     /* compute first N/2 rows of c */
      for(i=0;i<N/2;i++)
      for(j=0;j<N;j++){
           c[i][j]=0;
           for(k=0;k<N;k++)
             c[i][j]+=a[i][k]*b[k][j];
       }
       /* read the last N/2 rows from child */
       dread(1,&c[N/2][0],sizeof(c)/2));
     }
     else{
        /* child process */
       /* compute last N/2 rows of C */
```

भारतीय प्रौद्योगिकी संस्थान कानपुर INDIAN INSTITUTE OF TECHNOLOGY KANPUR

```
        for(i=N/2;i<N;i++)
        for(j=0;j<N;j++){
            c[i][j]=0;
            for(k=0;k<N;k++)
                c[i][j]+=a[i][k]*b[k][j];
         }
         /* send the partial result to parent */
          dwrite(1,&c[N/2][0],sizeof(c)/2);
         terminate(1); /* wait for next fork */
       }
}
```

The corresponding uniprocessor algorithm is given below.

```
int a[N][N],b[N][N],c[N][N];
main() {
 int i,j,k;
 for(i=0;i<N;i++)
 for(j=0;j<N;j++){
    c[i][j]=0;
    for(k=0;k<N;k++)
      c[i][j]+=a[i][k]*b[k][j];
 }
}
```

We studied the performance for different matrix order (N) and different communication modes. We measured the execution time of the problem in two ways. In the

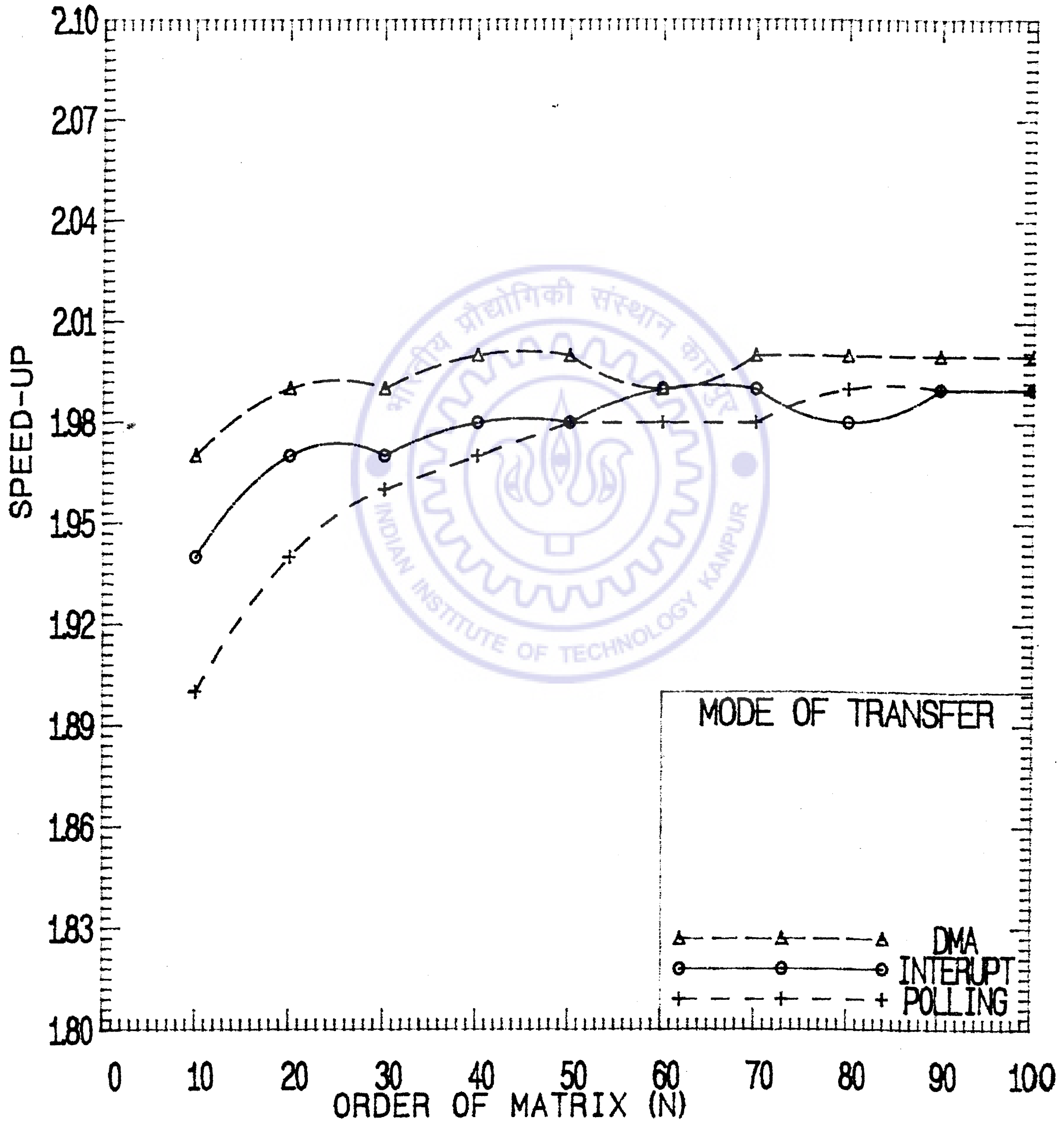


Fig. 6.1 SPEED-UP WITHOUT FORK DELAY

भारतीय प्रौद्योगिकी संस्थान कानपुर INDIAN INSTITUTE OF TECHNOLOGY KANPUR

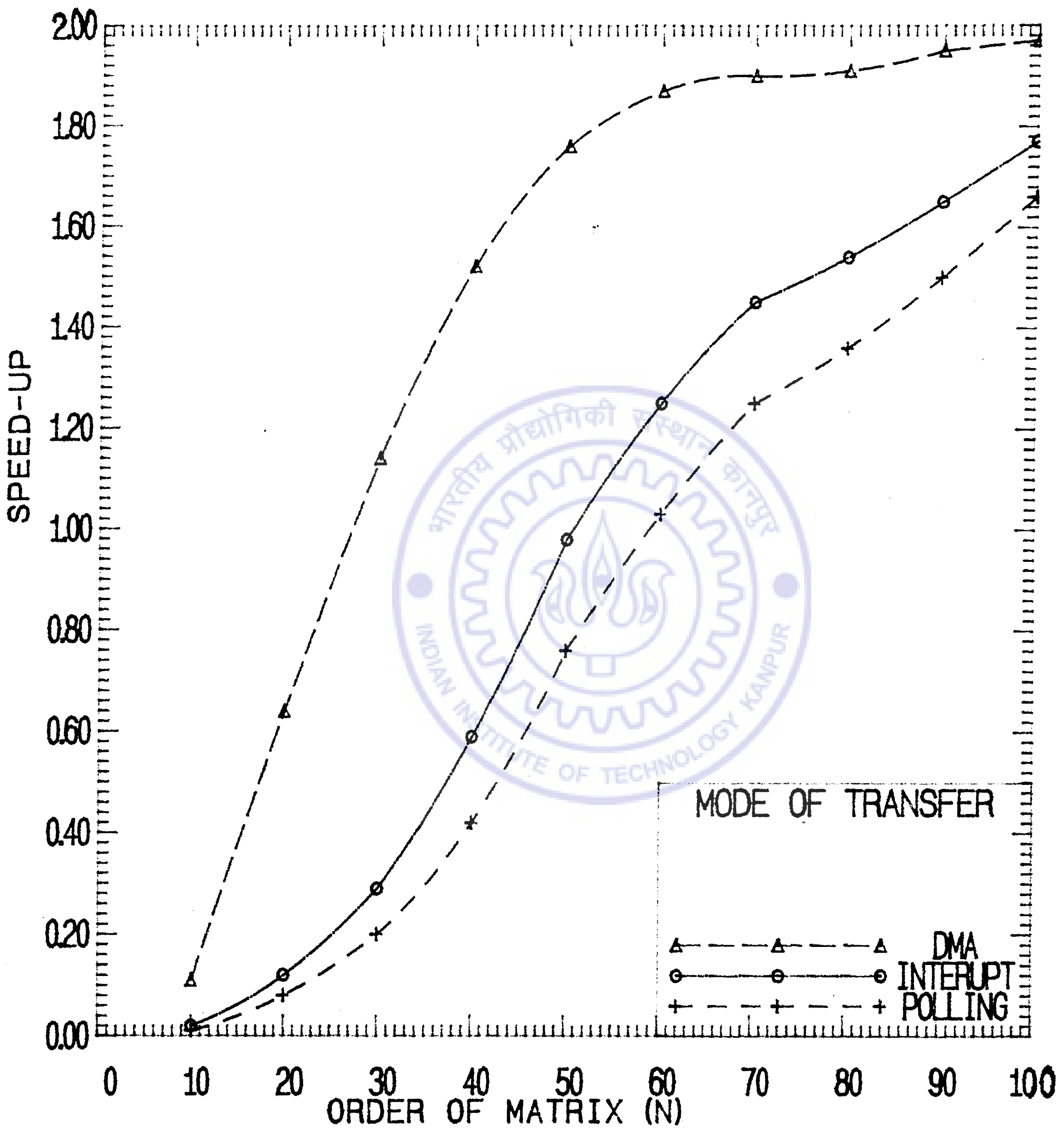


Fig. 6.2 SPEED-UP WITH FORK DELAY

भारतीय प्रौद्योगिकी संस्थान कानपुर
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

first step we ignored the time taken for remote process creation while measuring the execution time. We then observed the variations of the execution time for Polling and DMA mode of communication. This was carried out for different values of N. The result of this experiment is presented in figure 6.1. We observe that there is gain in speed for N greater than 25 in both the modes.

In the second step, we carried out the same experiment with due consideration of remote process creation time. The result of these experiments are summarised in figure 6.2. We see that there is no speed gain for N less 85 than for Polling method. However, we observe speed-up for N greater than only 25 in DMA mode. This clearly shows that the communication costs are much less in DMA mode than in software polling mode.

6.4 Conclusion

The performance analysis of KIMS shows that it is effective in solving different of problems with the expected speedup. Among the different communication modes supported in KIMS, the DMA mode gives best performance.

Chapter 7

Conclusion

In this chapter we summarize our experiences in developing KIMS multicomputer system. We also suggest some modifications which can be incorporated in future versions of KIMS for improving its performance.

7.1 Observations

The present configuration of KIMS consists of two AT's connected using the communication interface developed for this purpose. The system gives expected speedup.

During the testing of link interface card, some minor errors were detected in the printed circuit board connections. These errors were rectified using jumper wires. We have also incorporated these changes in the artwork of the PCB layout.

A small bug in the WARN interrupt generation circuit was also detected. The WARN interrupt, which indicates illegal link access, works for any illegal link access. But a WARN interrupt is also generated for the last valid read/write access of the links. The reason behind this is explained below.

The FIFO empty and FIFO full conditions of IDT 7203 FIFO change during the last valid read/write operation. This change occurs while the read/write process is still going on. This gives a false indication of illegal link access to the WARN interrupt

CENTRAL LIBRARY
I. I. T., KANPUR
Acc. No. A.114547

generation circuit, which generates an interrupt when the FIFO is empty (full) and a read (write) attempt is made.

In the current implementation, we have taken care of this problem through software, where the first WARN interrupt is ignored. However, the software solution implies some overhead and slows down the communication speed.

The hardware solution for this problem is simple as described below. It may be incorporated in future versions of communication interface cards.

Since the FIFO empty and FIFO full conditions change while the read / write operation is still going on, this causes triggering of a false WARN interrupt for the last valid read/write link access. This can be avoided, if the WARN interrupt generation circuit samples the FIFO full and FIFO empty condition only at the beginning of a machine cycle. The system interface of the PC/AT motherboard provides an ALE signal at the starting of every machine cycle. Thus, a latch can be used to sample the FIFO empty and FIFO full condition using ALE signal. However, the same FIFO status are also used for generation of DMA request. If the 8237 DMA controller finds that the DMA request signal is active throughout a DMA transfer, then it automatically starts the next transfer. So the DMA request has to be removed during the last valid DMA operation. Otherwise an invalid transfer will be made. Therefore, for the DMA request generation circuit, it is required to have the status of the FIFOs without synchronization with ALE. This demands that the latching of FIFO status should be done using a transparent latch. The latch will remain transparent during a DMA mode operations but sample the FIFO status using ALE during normal CPU cycles.

7.2 Suggestions for Future Modifications

In this section, we briefly state some modifications which can be incorporated in KIMS to obtain better performance.

- The WARN interrupt generation circuit should be modified in the way as described above.
- The current implementation of KIMS consists of only two CEs. The number of CEs should be increased for better performance. Moreover, for solving large scientific problems, it will be advantageous if the CPUs of the CEs are backed with additional arithmatic processors.
- The software environment provided is very primitive as far as the debugging of parallel algorithms are concerned. So a suitable debugger for this purpose will be certainly desirable.
- Currently, the language support is provided only in C language. To enable a larger class of users to use KIMS, language support in other commonly used languages like FORTRAN , Pascal, LISP should be provided.
- The current software environment does not provide automatic process mapping or parallelism detection. If these features are added then users with little knowledge about parallel processing will be able to use the system.

Appendix A

PAL Logic

As explained earlier, the entire control logic of the communication card is implemented using six 16L8 PALs. In this chapter we provide the details of each of these PALs. These PALs were programmed in CUPL PAL programming language. Listing of the CUPL programs corresponding to each PAL is given below.

Following conventions are followed in all the programs.

- Any signal name, which ends with a capital 'B', indicates that it is an active low signal.
- The symbol '#' stands for logical OR operation, whereas '&' stands for logical AND operation. Symbol '!' denotes logical NOT operation.

A.1 EPLD1 logic

The CUPL program listing of EPLD1 PAL is given below. This PAL is used in the *Address decoding block.*

```
Name        EPLD1;
Partno      1;
```

```
Date       30/4/92;
Revision   01;
Designer   BHASKAR;
Company    IITK;
Assembly   MMS Board;
Location   epld1;
Device     p1618;

/*************************************************************/
/* This device is used to decode A1..A15 and decide :-     */
/* linkB & genB                                            */
/*************************************************************/
/** Allowable Target Device Types : 1618 PAL,EP320      **/
/*************************************************************/
/**  Inputs  **/

PIN 1       = a1        ; /* CPU address A1 */
PIN 2       = a2        ; /* CPU address A2 */
PIN 4       = a3a4B     ; /* Nor of A3 & A4 from EPLD3*/
PIN 6       = a5        ; /* CPU Address A5 */
PIN 8       = a6        ; /* CPU Address A6 */
PIN 7       = a7        ; /* CPU Address A7 */
PIN 13      = a8        ; /* CPU address A8 */
PIN 14      = a9        ; /* CPU address A9 */
PIN 15      = a10       ; /* CPU address A10*/
PIN 16      = a11       ; /* CPU address A11*/
PIN 17      = a12       ; /* CPU address A12*/
PIN 5         = a13        ; /* CPU Address A13*/
PIN 18        = a14        ; /* CPU address A14*/
PIN 3         = a15        ; /* CPU address A15*/
PIN 9         = aen        ; /* AEN indicates  */
                             /* valid Address  */
PIN 11        = dackB      ; /* Ack. of DMA channel 3*/

/**  Outputs  **/

PIN 12        = linkB    ; /* Link Select Bar  */
PIN 19        = genB     ; /* General Regis. Select Bar */

/* Declarations and Intermediate Variable Definitions */

Field memadrH = [a15..a5]  ; /*  The High Addresses      */
Field memadrL = [a2..a1]   ; /*  The Low Address         */
DMAactive = (aen & !dackB); /*  Indicates active DMAC */

linkaddr  = !aen & memadrH:140 & a4a4B & memadrL:146;
/* ^---- Indicates valid link address                 */

genaddr=!aen & memadrH:140 & a4a4B & memadrL:[140..146];
/* ^---- Indicates valid general register address     */

/**  Logic Equations  **/

linkB = !( linkaddr # DMAactive) ;
```

```
genB  = !( genaddr ) ;
```

A.2 EPLD2 logic

The CUPL program listing of EPLD2 PAL is given below. This PAL used in the *data bus interface block.*

```
Name        EPLD2;
Partno      2;
Date        30/4/92;
Revision    01;
Designer    BHASKAR;
Company     IITK;
Assembly    MMS Board;
Location    epld2;
Device      p1618;

/***********************************************************/
/* This device is used to decode switch sw0,sw1 and sig-*/
/* nals tmpillrdB,linkB,genB and decides the 4 buffers' */
/* output enable signals sg0B .. sg3B(active low)        */
/***********************************************************/
/** Allowable Target Device Types : 1618 PAL,EP320      **/
/***********************************************************/
```

```
/**  Inputs  **/

PIN 9         = linkB     ; /* link select bar         */
PIN 15        = genB      ; /* general reg select bar */
PIN [3..4]    = [sw0..1]  ; /* Card select switches    */
PIN 7         = maskor    ; /*  OR of mask0..mask3     */
PIN 8         = tmpillrdB ; /* temp illegal read  bar */
  /* ^--this signal indicates >1 link selection        */

PIN 1         = iordB     ; /* IORD bar from CPU       */
PIN 2         = iowrB     ; /* IOWR bar from CPU       */
PIN 14        = cntrlsel  ; /* cntrol register select */
                            /*  generated by EPLD3     */
/**  Outputs  **/

PIN 19        = sg0B    ;    /* enable DBF0              */
PIN 18        = sg1B    ;    /* enable DBF1              */
PIN 17        = sg2B    ;    /* enable DBF2              */
PIN 16        = sg3B    ;    /* enable trancv3           */

/* Declarations and Intermediate Variable Definitions */

Field cardadr = [sw1..0];    /* Card No. switches      */

legalrd= tmpillrdB & !iordB & maskor & !linkB ;
/* ^--indictes not more than one link being accessed  */
```

INDIAN INSTITUTE OF TECHNOLOGY KANPUR
प्रौद्योगिकी संस्थान

```
legalwr= iordB & !linkB ;
/* ^--indicates link write access                    */

validlink = legalrd # legalwr;
/* ^--indicates valid link access                    */

/**  Logic Equations  **/

sg0B  = !( validlink # cntrlsel # (cardadr:00 & !genB) );
sg1B  = !( validlink # cntrlsel # (cardadr:01 & !genB) );
sg2B  = !( cardadr:02 & !genB & !cntrlsel);
sg3B  = !( cardadr:03 & !genB & !cntrlsel );
```

A.3 EPLD3 logic

The CUPL program listing of EPLD3 PAL is given below. This PAL used in the *general register block*.

```
Name      EPLD3;
Partno    3;
Date      30/4/92;
Revision  01;
Designer  BHASKAR;
Company   IITK;
Assembly  MMS Board;
Location  epld3;
```

```
Device    p1618;

/*************************************************************/
/* This device is used to generate rd/wr signals for        */
/* the general purpose registers resetB,cntrlw,maskw         */
/* stat1rdB,stat2rdB                                         */
/*************************************************************/
/** Allowable Target Device Types : 1618                   **/
/*************************************************************/
/**  Inputs  **/

PIN 9       = linkB     ; /* link select bar          */
PIN 6       = genB      ; /* general reg select bar   */
PIN 1       = iordB     ; /* IORD bar from cpu/dmac   */
PIN 2       = iowrB     ; /* IOWR bar from cpu/dmac   */
PIN 8       = reset     ; /* RESET DRV from cpu       */
PIN 7       = a0        ; /* address A0..A4 from cpu */
PIN 11      = a1        ;
PIN 4       = a2        ;
PIN 5       = a3        ;
PIN 3       = a4        ;


/**  Outputs  **/

PIN 13      = resetB  ;    /* reset bar for fifos          */
PIN 15      = cntrlw  ;    /* cntrl register write NOBAR */
PIN 14      = maskw   ;    /* mask register write  NOBAR */
PIN 18       = stat1rB ;     /* status1 register read bar   */
PIN 19       = stat2rB ;     /* status2 register read bar   */
PIN 16       = iocs16B ;     /* iocs16 bar to indicate 16   */
                             /* 16 bit register operation   */
PIN 17       = a3a4B   ;     /* NOR of A3 and  A4           */
PIN 12      = cntrlsel;     /* Control Register select    */


/** Declarations and Intermediate Variable Definitions **/


Field addr = [a2..0] ; /* Give The Address Bus      */



/**  Logic Equations  **/


resetB    = !(reset # (!genB & !iowrB & addr:04) );
cntrlw    = (!genB & !iowrB & addr:02 );
maskw     = (!genB & !iowrB & addr:0);
stat1rB   = !(!genB & !iordB & addr:2);
stat2rB   = !(!genB & !iordB & addr:4);
iocs16B   = !(!genB # !linkB) ;
a3a4B     = !(a3 # a4);
cntrlsel = (!genB & addr:02);
```

A.4 EPLD4 logic

The CUPL program listing of EPLD4 PAL is given below. This PAL used in the *link interface.*

```
Name      EPLD4;
Partno    4;
Date      30/4/92;
Revision  01;
Designer  BHASKAR;
Company   IITK;
Assembly  MMS Board;
Location  epld4;
Device    ep320;

/*************************************************************/
/* This device is used to generate rd/wr signals for        */
/* the link fifos.                                           */
/*************************************************************/
/** Allowable Target Device Types : 1618 PAL,EP320        **/
/*************************************************************/
/**  Inputs  **/

PIN 3        = linkB      ; /* link select bar    */
PIN 11       = illwr      ; /* illegal write      */
PIN 9        = tmpillrdB  ; /* temp illegal read */
PIN 1        = iordB      ; /* IORD bar from cpu */
PIN 2          = iowrB        ; /* IOWR bar from cpu */
PIN [5..8]     = [mask0..3]   ; /* mask0.. mask3     */


/**  Outputs  **/

PIN 19         = link0rdB   ;  /* link-0 read bar   */
PIN 18         = link0wrB   ;  /* link-0 write bar  */
PIN 17         = link1rdB   ;  /* link-1 read bar   */
PIN 16         = link1wrB   ;  /* link-1 write bar  */
PIN 12         = link2rdB   ;  /* link-2 read bar   */
PIN 13         = link2wrB   ;  /* link-2 write bar  */
PIN 14         = link3rdB   ;  /* link-3 read bar   */
PIN 15         = link3wrB   ;  /* link-3 write bar  */


/**  Logic Equations  **/

link0rdB = !( (!linkB & mask0 & !iordB) & tmpillrdB);
link0wrB = !(!linkB & mask0 & !iowrB) ;
link1rdB = !( (!linkB & mask1 & !iordB) & tmpillrdB);
link1wrB = !(!linkB & mask1 & !iowrB);
link2rdB = !( (!linkB & mask2 & !iordB) & tmpillrdB);
link2wrB = !(!linkB & mask2 & !iowrB);
link3rdB = !( (!linkB & mask3 & !iordB) & tmpillrdB);
link3wrB = !(!linkB & mask3 & !iowrB);
```

A.5 EPLD5 logic

The CUPL program listing of EPLD5 PAL is given below. This PAL used in the *inter-card communication interface.*

```
Name      EPLD5;
Partno    5;
Date      30/4/92;
Revision  01;
Designer  BHASKAR;
Company   IITK;
Assembly  MMS Board;
Location  epld5;
Device    ep320;


/************************************************************/
/* This device is used to generate some temporary signals */
/* which are then used in epld6 to generate differnet     */
/* interrupts,dma request etc..                           */
/************************************************************/
/** Allowable Target Device Types : 1618                  **/
/************************************************************/
/**  Inputs  **/

PIN 6       = feB3        ;/* FIFO empty status of link 3  */
PIN [7..9]  = [feB0..2]   ;/* FIFO empty status of link0..2*/
PIN 5       = ffB3        ;/* FIFO full status of link 3   */
PIN [2..4]  = [ffB0..2]   ;/* FIFO full status of link0..2 */
```

भारतीय प्रौद्योगिकी संस्थान कानपुर INDIAN INSTITUTE OF TECHNOLOGY KANPUR

```
PIN 1         = prev       ;/* PREV from previous card i-1  */
PIN 11        = mask3      ;/* mask bit 3 of mask reg       */
PIN [17..15]  = [mask0..2];/* mask bits 0..2  of mask reg   */
PIN 14        = tmpillrdB ;/* intermediate illegal read     */
/* ^--this is output also and denotes selection >1 link     */


/**  Outputs  **/


PIN 19        = newdataB  ;/* arrival of data in sel. link */
PIN 13        = locillrdB ;/* indicates local illegal read */
PIN 12        = locillwrB ;/* denotes local illeag write   */
PIN 18        = maskor    ;/* logical OR of m0..m3    */



/** intermediate variables **/


 illmask0 =((mask0 & mask1)#(mask0 & mask2)#(mask0 & mask3));
 illmask1 =((mask1 & mask2)#(mask1 & mask3));
 illmask2 =(mask2 & mask3 );


 enbfifemp = (mask0&!feB0)#(mask1&!feB1)#(mask2&!feB2)#(mask3&!feB3);
 /* ^--denotes >=1  enabled FIFO in card is  empty */


 enbfiffull=(mask0&!ffB0)#(mask1&!ffB1)#(mask2&!ffB2)#(mask3&!ffB3);
 /* ^--denotes >=1 enabled FIFO in card is    full */


/**  Logic Equations  **/
```

```
tmpillrdB = !(illmask0#illmask1#illmask2#(prev&maskor));

locillrdB = !(!tmpillrdB # enbfifemp);

locillwrB = !enbfiffull;

maskor     = mask0 # mask1 # mask2 # mask3 ;

newdataB   =!((mask0&feB0)#(mask1&feB1)#(mask2&feB2)#(mask3&feB3));
```

A.6 EPLD6 logic

The CUPL program listing of EPLD6 PAL is given below. This PAL used in the *interrupt and DMA control block.*

```
Name       EPLD6;

Partno     6;

Date       30/4/92;

Revision   01;

Designer   BHASKAR;

Company    IITK;

Assembly   MMS Board;

Location   epld6;

Device     p1618;


/**************************************************************/

/* This device is used to generate --next,int1,int2, drq */

/* illioint,datacameint                                */

/**************************************************************/
```

```
/** Allowable Target Device Types : 1618,ep320          **/
/*************************************************************/
/**  Inputs  **/

PIN 3  = linkB        ;/* link select         */
PIN 9  = illrd        ;/* illegal read access */
PIN 11 = illwr        ;/* illegal write access*/
PIN 4  = maskor       ;/* mask OR from epld5  */
PIN 5  = dav          ;/* data  has arrived   */
PIN 16 = dmaenbl      ;/* DMA enable of cntrl */
PIN 18 = direction    ;/* DIRecton of cntrl   */
PIN 8  = illintenbl   ;/* WARN enable of cntrl*/
PIN 15 = davintenbl   ;/* DFAV enble of cntrl */
PIN 1  = iordB        ;/* cpu/DMAC iord signl */
PIN 2  = iowrB        ;/* cpu/DMAC iowr signal*/
PIN 6  = prev         ;/* PREV from prev card */
PIN 7  = genB         ;/* general reg. select */

/**  Outputs  **/

PIN 12 = nextB       ;/* NEXT signal        */
PIN 14 = drq         ;/* DMA request(DRQ3)  */
PIN 17 = int1        ;/* WARN interrupt     */
PIN 19 = int2        ;/* DAV interrupt      */
PIN 13 = illrdwrB    ;/* illegal read or write   */

/**  Logic Equations  **/
```

```
nextB = !(maskor # prev);
drq = ( dmaenbl&((direction&!illrd)#(!direction & !illwr)));
drq.oe = dmaenbl;
illrdwrB =!((illrd&!iordB&!linkB)#(illwr&!iowrB&!linkB));
int1.oe = illintenbl;
int1 = !illrdwrB #(int1 & genB);
int2.oe = davintenbl;
int2       = dav;
```

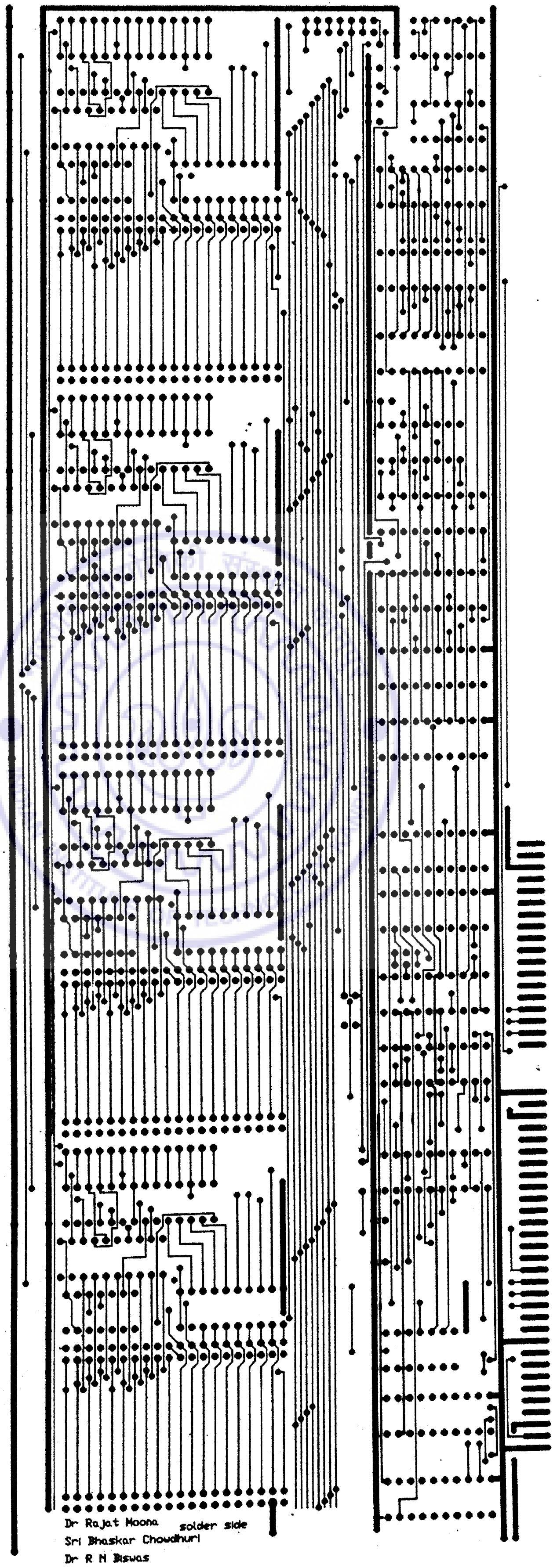
भारतीय प्रौद्योगिकी संस्थान कानपुर INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Appendix B

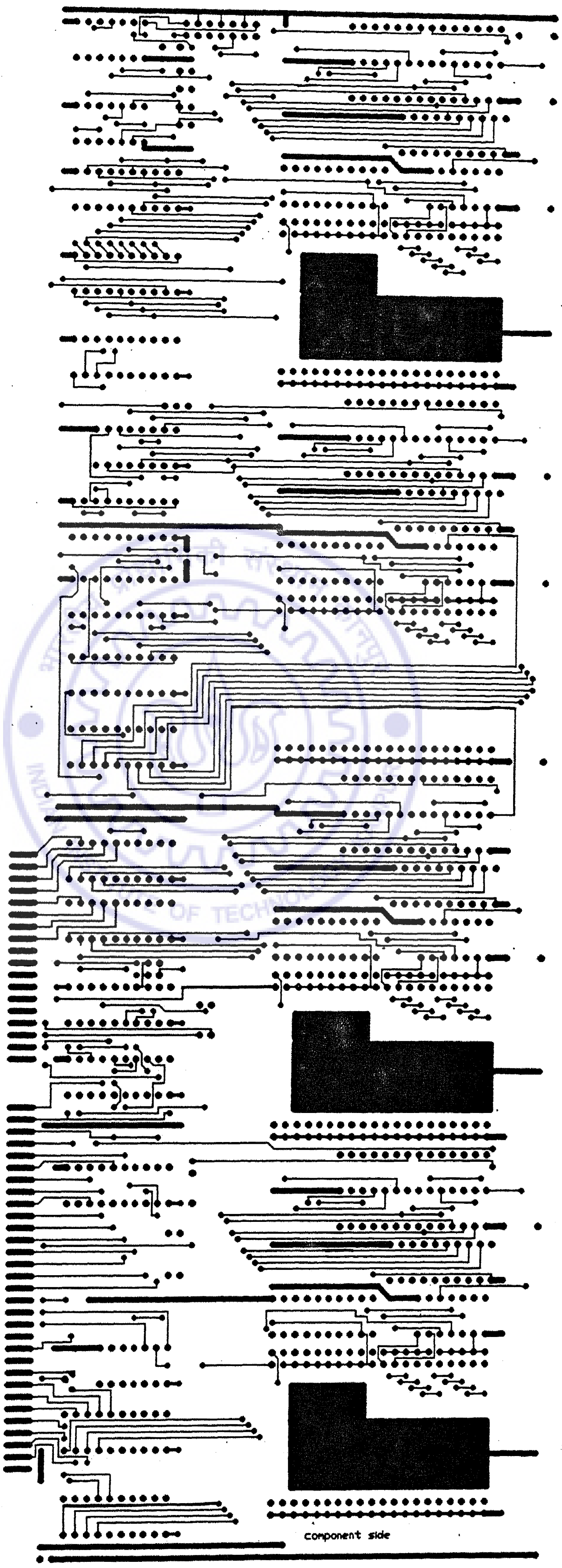
PCB Layout of the extension card

The PCB layout of the communication card was designed using *Protel* PCB desin tool. The card is realised on a double sided PCB with plated through holes. The detailed layouts of the two sides of the card are shown below.

प्रौद्योगिकी संस्थान
INDIAN INSTITUTE OF TECHNOLOGY KANPUR



Dr Rajat Moona
Sri Bhaskar Chowdhuri
Dr R N Biswas


component side

Bibliography

[1] S. R. Das, N. H. Vaidya, and L. M. Patnaik. "Design and implementaion of a hypercube multiprocessor". *Microprocessors and Microsystem*, 14(2), March 1990.

[2] S. K. Ghosal, S. Guha, and V. Rajaraman. "Simple low-cost multiprocessor based on message passing FIFO links". *Microprocessors and Microsystem*, 14(5), June 1990.

[3] D. M. Goodeve and R. W. Taylor. "Communications coprocessor for the Acorn RISC machine". *Microprocessors and Microsystem*, 14(5), June 1990.

[4] R. Moona and V. Rajaraman. "Design and implementation of a broadcast cube multiprocessor". In *Proc. KBCS conf. on Knowledge Based Computer Systems*, 1989.

[5] R. Moona and V. Rajaraman. "A FIFO-based multicust network and its use in multicomputers". *Microprocessors and Microsystem*, 15(10), December 1991.

[6] R. Moona and V. Rajaraman. "Multidimensional multilink multicomputer: A general purpose parallel computer". *Journal of Indian Institute of Science*, 71(2), 1991.

[7] V. L. Narasimhan, R. J. Coote, and J. Rission. "Design and analysis of a loosely coupled dual processor system". *Journal of Microcomputer Applications*, 15(2):121–136, 1992.

[8] Charles L. Seitz. "The Cosmic Cube". *Communications of the ACM*, 28(1):22–33, 1985.

A114547

Th
001.64404
C459d

A114547

Date Slip

This book is to be returned on th
date last stamped.

CSE-1992-M-CHO-DES