

DESIGN OF ADSP 2100 - BASED
MULTI-PROCESSOR ARRAY

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
Master of Technology



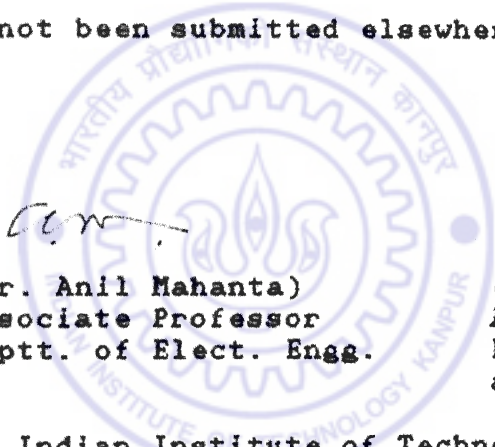
by
MIRZA MOHD. SUFYAN BEG

to the
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
March 1994


11-3-94
M. Sufyan Beg

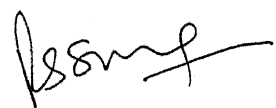
Certificate

It is certified that the work contained in the thesis entitled "Design of ADSP 2100-Based Multiprocessor Array", by Mirza Mohd. Sufyan Beg, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.



March 1994


(Dr. Anil Mahanta)
Associate Professor
Deptt. of Elect. Engg.


(Dr. Rajat Moona)
Assistant Professor
Deptt. of Comp. Science
and Engineering

Indian Institute of Technology Kanpur
Kanpur 208 016



13 APR 1994

CENTRAL LIBRARY
IIT KANPUR

Acc. No. A. 117680

TH
621-3670/51
B393d

EE-1994-M-BEG-DES

ABSTRACT

The real-time signal and image processing applications make the parallel processing architectures inevitable. Here a one-dimensional multiprocessor array architecture using ADSP 2100 DSP chips as processing elements (PEs) is proposed and designed. The processor array is interfaced to PC-AT. There is a broadcast channel which is used for down/up-loading of program and data to the PEs. Other two channels have been provided for interprocessor communication. One of them, namely X-channel, is also used for systolically passing input data to the PEs. Additional hardware has been included to facilitate the array with single-cycle multiple-destination data transfer capabilities-the facility lacked by DSP chips. This facility enables the matching of communication bandwidth to the computational throughput of the processor. The design has been accomplished upto the PCB level, part of the system has been assembled and partial testing has also been carried out.

ACKNOWLEDGEMENTS

It is due to the grace of Allah s.w.t., the good wishes of my parents and elders, and of course the pains taken by my teachers that I have been able to complete this thesis entitled "Design of ADSP 2100-Based Multiprocessor Array".

I am highly indebted to Dr. Anil Mahanta, who guided me through this thesis with intense academic temper and utmost affection. He has the rare quality of coming from his great heights down to drag me along with him up the steep slope of knowledge. Thank you Sir, for being so much considerate to me.

I also owe sincere gratitude to Dr. Rajat Moona who, by virtue of his deep knowledge and sheer experience, set some excellent guidelines for me. He taught me some very fine points in the art of hardware design. Thank you so much, Sir.

My special thanks to Mr. Taj Alam and his family members, Nazrul, Satyamji, Rajivji, Gulab, Tufail Sb. and all those who made my stay at IIT Kanpur a pleasant and a memorable experience. I won't thank Khurshid because, after all, a friend in need is a friend indeed.

CONTENTS

1	Introduction.....	1
1.1	Characteristics of Signal and Image Processing Algorithms.....	2
1.2	Parallel Architectures.....	4
1.3	Objective of the Thesis.....	5
1.4	Organization of the Thesis.....	5
2	Architectures for Signal and Image Processing Applications	6
2.1	Characteristics of Low-Level Image Processing and Signal Processing Algorithms.....	6
2.2	Approaches to Parallel Architectures for Image and Signal Processing.....	8
2.2.1	SIMD Arrays.....	8
2.2.2	MIMD Arrays.....	9
2.2.3	VLSI Arrays.....	9
2.2.3.1	Systolic Arrays.....	9
2.2.3.2	Wavefront Arrays.....	11
2.2.4	Summary.....	11
2.3	Architectural Decisions.....	12
2.3.1	Array Topology.....	12
2.3.2	Processing Element Architecture.....	13
3	Overview of Proposed Architecture.....	16
3.1	Architecture of ADSP 2100.....	16
3.2	Interprocessor Communication Links.....	18
3.3	Proposed Architecture.....	19
3.3.1	Types of Data Transfers Supported.....	21

	3.3.2	Functioning of the Array.....	23
	3.3.3	Interface Card.....	24
	3.3.4	Processing Element Card.....	26
	3.4	Modes of Operation of the Processor Array.....	26
4		Design of Host Interface And Backplane.....	28
	4.1	I/O Port Select Generation.....	29
	4.2	Control Register.....	29
	4.3	Status Register.....	31
	4.4	Loading of XFIFO.....	32
	4.5	Data, Address and Control Buffers.....	32
	4.6	Interrupt Circuitry.....	33
	4.7	Interface Card Assembly.....	33
5		Design of Processing Element.....	41
	5.1	Design of Interface to Global Channel	43
	5.2	Design of Single-Cycle Multiple-Destination Data Transfers.....	47
	5.3	Run-Time Flow Control.....	51
	5.3.1	X-Channel Flow Control.....	51
	5.3.2	Y-Channel Flow Control.....	54
	5.4	Processing Element Card Assembly.....	55
6		Discussions and Current Status of the Work.....	60
		References.....	64
		Appendix A.....	66
		Appendix B.....	70
		Appendix C.....	71
		Appendix D.....	86

CHAPTER 1

INTRODUCTION

Digital Signal and Image Processing operations are usually computationally intensive, because of the huge amount of data that must be processed and of the complexity of the elementary operations involved. Typically the image size varies from 256 X 256 to 1024 X 1024, and so it means that about 3 Mbytes of data has to be processed for a single colour image. And when it comes to applications like digital video processing or robotic vision, all these calculations must be performed in real-time, i.e. at a rate of 25 to 30 images per second. A computation rate in excess of a billion operations per second may frequently become necessary for real-time performance [1]. For instance, in linear operations like spatial filtering, convolution and edge detection, the necessary throughput rate is as high as 10^2 to 10^5 MOPs. The minimum required throughput climbs up to about 10^3 to 10^7 MOPs for second order operations like sorting operations, median filtering and nearest-neighbour classification. The higher order operations such as spectral processing, adaptive operations and the matrix based operations require a terrific 10^4 to 10^8 MOPs throughput. Based on the speed requirement, the computational complexity and the data volume, it can be seen that the limit of general purpose

computers can easily be reached in digital signal processing applications. Parallel processing provides the mainstream solutions to fast image processing and computer vision.

Although computationally intensive, signal and image processing algorithms involve a relatively small set of core operations which are repetitive and regular in nature, requiring only local communication. Architecture of the processor can, therefore, be optimized for efficient execution of these operations at the expense of the more general features offered by a general-purpose computing system but not required in these applications. For scientific computations and real-time signal processing applications, special-purpose processor arrays like systolic/wavefront arrays have been found to offer a cost-effective solution [1].

1.1 Characteristics of Signal and Image Processing Algorithms :

Parallelism suits well to the tasks of signal processing and to the nature of digital images [2]. Following are the various types of parallelism that can be identified in image processing :

- * *Geometrical parallelism*
- * *Neighbourhood parallelism*
- * *Pixel-bit parallelism*
- * *Operator parallelism*

As the digital images are usually sampled on a rectangular grid and are stored as a 2-D array, they possess an inherent geometrical parallelism. This parallelism can be exploited by employing a large 2-D array of processors, preferably with

one-processor-per-pixel configuration. For large image size, however, this becomes impossible, the current technological standards being the limit. As a compromise, segments (either squares or strips) of the image are assigned to each processor. This image split might create the problem of border effects.

Many digital image processing algorithms are essentially neighbourhood operations of the form :

$$Y_{ij} = F(x_{i+r, j+s}) \quad (r, s) \in A$$

where x_{ij} , y_{ij} are the input and output images respectively, F is an operator (linear or nonlinear) and A is its processing window. This parallelism denotes the parallel execution of local neighbourhood operations. In this case, a local processor must have access or communication to its neighbours' data.

Pixel-bit parallelism exploit the fact that an image can be decomposed into b bit planes, where b is the number of bits in the image pixel (usually $b=1$ or 8). Several image processing operations, notably the linear operations, can be performed on each bit plane independently. The arithmetic that is performed on bit planes is called *distributed arithmetic*.

Two types of operator parallelisms exist in image processing operations : *pipelining* and *parallel decomposition*. Pipelining is the most commonly used and can be expressed as :

$$Y = F(X) = F_n \cdot (F_{n-1} (\dots F_2 (F_1 (X)) \dots))$$

where X is the input image or image subregion, Y is the output image, F is an operator and F_i , $i=1, \dots, n$ are its cascade decomposition. Typical example of pipelining is the cascade realization of linear digital filters.

Parallel Decomposition involves operators of the form :

$$Y = F(X) = F_1(X) \parallel F_2(X) \parallel \dots \parallel F_n(X)$$

where \parallel denotes parallel execution. A typical example of this type of parallelism is the parallel realization of 2-D digital filters.

1.2 Parallel Architectures:

Parallel Architectures can be classified in two broad categories : *Single Instruction Multiple Data (SIMD)* and *Multiple Instruction Multiple Data (MIMD)* machines [2]. SIMD computers consist of arrays of simple processing elements, which are connected to their immediate neighbourhood to form a processor grid. They exploit both geometrical and neighbourhood parallelism at the same time. Instructions are broadcasted by the host to all PEs. These are ideally suited for low-level vision applications.

MIMD machines for signal processing applications have had a widespread use in the last decade (Ref to Ch.2). They can be divided in two large classes : *distributed memory* and *common memory machines*. In the case of common memory architectures, the images are stored in the common memory and can be accessed by any processor anytime. They suffer from conflicts in memory access. In the case of distributed memory architectures, each processor has its own local memory and communicates to the other by using communication links and/or a common bus. Communication by parallel or serial links is widely used in machines based on transputers or on certain Digital Signal Processors (Ref. to Ch.2). The mixed topologies having both communication links (for message passing) and a high speed common bus (for image transfer) are preferable

for such applications .

1.3 Objectives of the Thesis:

(a) To design and implement a multiprocessor array using DSP microprocessor (Analog Devices ADSP 2100) as the processing element (PE) and interface the array to a host computer (PC-AT/386 in this case).

(b) To map a few signal processing algorithms onto this array and evaluate its performance.

1.4 Organization of the Thesis :

Chapter 2 gives a general view of the architectural requirements for signal and image processing applications. It also briefly reviews the various approaches and the consequent existing architectures customized for signal processing domain.

The proposed architecture of this thesis is then introduced in Chapter 3. The different modes of operation of our architecture are also discussed.

In Chapter 4, the detailed design of the PC-plug-in card that interfaces the proposed multiprocessor array to the host is dealt with.

Chapter 5 discusses in detail the various issues involved in the design of the processing element (PE) card. This includes the design of the interprocessor communication links and the broadcasting facility from the host to the PEs.

In chapter 6, we discuss the initial testing steps that have been carried out and give the present status of the work.

CHAPTER 2

ARCHITECTURES FOR SIGNAL AND IMAGE PROCESSING APPLICATIONS

Very high performance computer systems must rely heavily on parallelism, since there are several physical and technological limits on the ultimate speed of any single processor. The challenge with parallel system is to distribute the processing load, programs and data, among the PEs in such a balanced way that the resources remain active during as much time as possible. The two main paradigms of programming parallel systems are *data parallelism* and *task parallelism*.

In data parallelism, the data to be processed is distributed among the PEs all of which execute the same program. With task parallelism, an algorithm is split into sub-tasks which are assigned to different PEs and run concurrently. As the maximum number of sub-tasks found in a single algorithm is limited, the task parallelism is difficult to apply when the number of processors grows. Most often it is the data parallelism that is more adequate for implementing low-level image and signal processing algorithms on parallel computing systems [2].

2.1 Characteristics of Low-Level Image Processing and Signal Processing Algorithms :

Most low-level image processing algorithms have some common

features which influence the choice of image processing parallel systems [2]. As an example, most of these algorithms are *context-intensive* and *position-invariant*. The first feature means that the algorithm's features are fixed beforehand independently of local image features. The second feature makes the processing independent of the location of the image pixels. Moreover, the image is represented by a regular structure. These characteristics influenced the first generation of specific parallel image processing SIMD (Single Instruction Multiple Data) and pipeline architectures. Recent technological advances have led to the development of more powerful and flexible processors, some of which have been adapted to MIMD (Multiple Instruction Multiple Data) architectures, thus spreading the use of distributed memory architectures.

Low-level image processing operators can be classified loosely as *local*, *point-wise* and *global* operators [2]. With local operators, the value of a processed pixel depends only on its value and on the values of the pixels placed in its neighbourhood. They generally have a square or rectangular geometry. Examples are image convolution (with local kernels) and image correlation. The implementation of local operations on parallel computing structures is usually done by splitting the images into sub-images which are distributed by the PEs.

With point-wise operators, the value of a processed pixel depends only on the original value of that pixel, or on the value of that pixel in different images. Useful point-wise operators are the difference between time-sequenced images (for movement

detection or inter-frame coding) and the modification of the image quantization levels. Parallel point-wise algorithms do not require any communication between PEs. Their simplicity leads to the direct implementation onto the hardware or firmware in a large number of image processing systems.

A global operation uses the information spread in the image pixel array to compute the value of each pixel of the processed image. Histogram equalization, image coordinate transformations and component labeling are some examples of global operations usually applied in low and intermediate level image processing.

2.2 Approaches to Parallel Architectures for Image and Signal Processing :

Various architectures have been proposed for parallel implementation of image and signal processing algorithms using the concepts of pipelining and multiprocessing. SIMD arrays, MIMD arrays and special-purpose VLSI arrays are representative of various attempts at solving parallel processing problems.

2.2.1 SIMD Arrays :

SIMD computers (Fig 2.1) are implemented as an array (usually 2-D) of locally-connected processors each with its local memory. The host broadcasts the instructions to all the processors, and all of them execute the same instruction simultaneously. Examples of SIMD machines are the ILLIAC IV system, NASA's Massively Parallel Processor (MPP) and ICL's Distributed Array Processor (DAP) [1].

2.2.2 MIMD Arrays :

MIMD computers consist of a number of PEs, each with its own control unit, program and data (Fig 2.2). For the purpose of increasing processing parallelism, the overall processing task may be distributed among the PEs. There may result a decreased throughput when multiple PEs try to access the shared memory simultaneously. Nevertheless, the flexibilities in MIMD structures are often essential in order to deal with irregularly structured algorithms, such as those appearing in intelligent image processing and vision analysis applications [1].

2.2.3 VLSI Arrays :

A new approach based on VLSI array processor is becoming increasingly competitive. These arrays maximize the strength of VLSI in terms of intensive and pipelined computing and yet circumvent its main limitation on communication. As illustrated in Fig 2.3, the massive concurrency in VLSI arrays such as systolic/wavefront arrays is derived from pipeline processing, parallel processing, or both [1].

2.2.3.1 Systolic Arrays :

In systolic arrays, data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory [3]. To implement a variety of computations, data flow in a systolic system may be at multiple speeds in multiple directions - both inputs and (partial) results flow, whereas only results flow in classical pipelined systems.

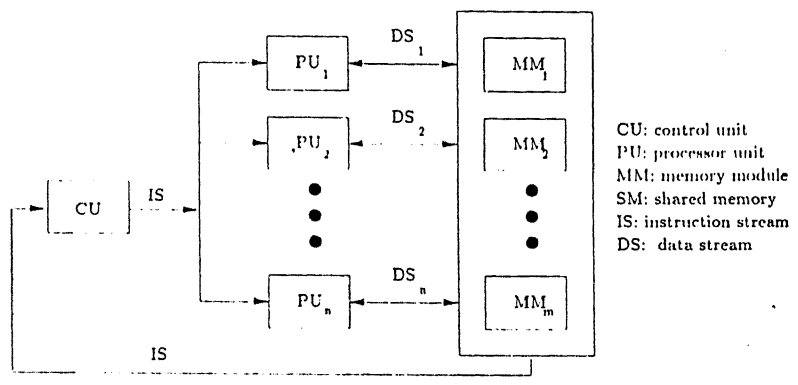


Fig. 2.1 SIMD Architecture.

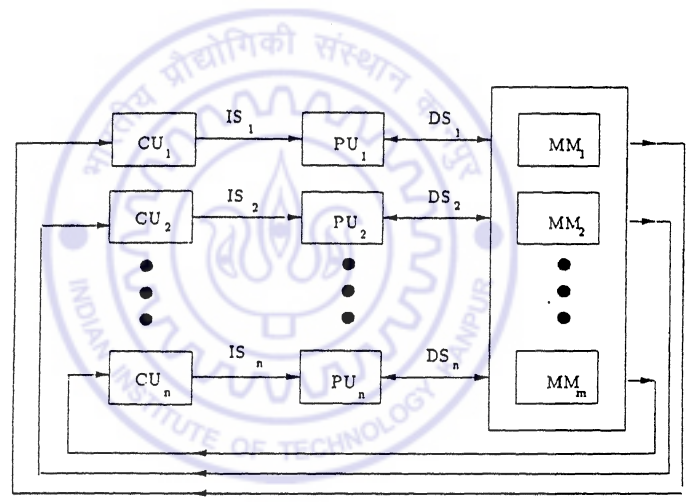
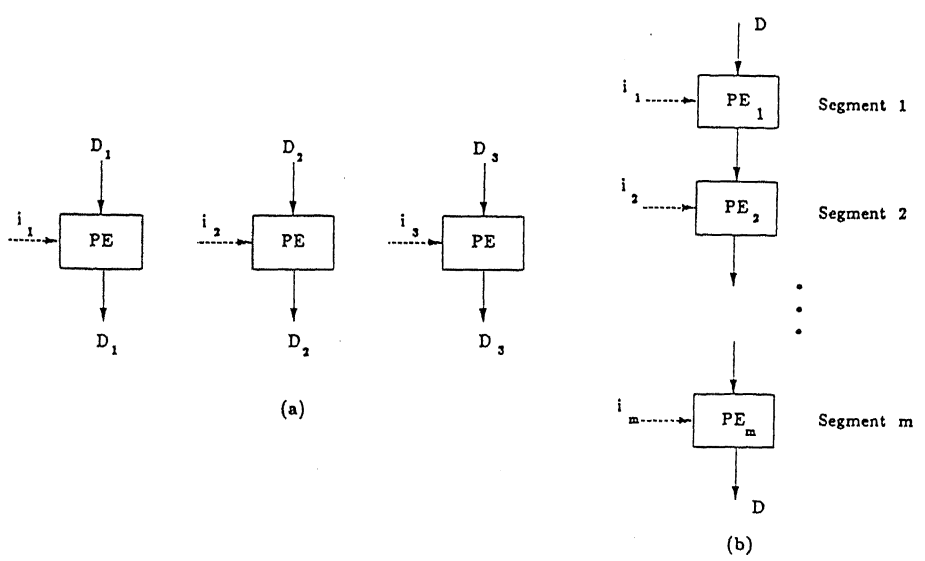


Fig. 2.2 MIMD Architecture.



Generally speaking, a systolic system is easy to implement because of its regularity and easy to reconfigure (to meet various outside constraints) because of its modularity. The Warp computer at CMU is an operational systolic array [4]. It is a one-dimensional array of powerful, programmable processing elements and is capable of efficiently implementing both data-parallel and purely systolic computation.

2.2.3.2 Wavefront Arrays :

The major feature distinguishing the wavefront array from the systolic array is the data driven property, i.e. there is no global timing reference in the wavefront array [1]. The information in the wavefront array is transferred by mutual convenience between a PE and its immediate neighbours by means of a simple handshaking protocol. The wavefront array processors possess most of the advantages of the systolic arrays, such as extensive pipelining and multiprocessing, regularity and modularity. More significantly, it also possesses the asynchronous data-driven capability of data flow machines and can, therefore, accommodate the critical problem of timing uncertainty in VLSI array systems.

2.2.4 Summary :

A critical survey of image processing algorithms and architectures has been made by Cypher et al [5]. It reveals that mesh connected SIMD computer is clearly very efficient at performing local neighbourhood operations. It is, however

inappropriate for performing high-level computer vision tasks as it is inefficient when data has to be moved long distances in the pointer-based communication. The pyramid computer seems to be comparable in power to a mesh connected computer that has been augmented with a tree of processors. The hypercube, on the other hand is better suited to high-level computer vision. The hypercube with independent communication is found to be the strongest model of the parallel computers studied. However, it is more difficult to build than the other types of parallel computers. The hypercube with SIMD communication and the shuffle-exchange and cube-connected cycles computers, are less expensive options that also perform long distance communication efficiently. However, these computers are less efficient than the mesh and pyramid computers at performing neighbourhood operations.

2.3 Architectural Decisions :

In designing a multiprocessor array for signal processing and low-level image processing applications, two crucial architectural decisions have to be made at the outset :

- * array topology (including interconnection scheme)
- * architecture of the processing element.

2.3.1 Array Topology :

Among the various topologies proposed [5], it has been found that 1-D and 2-D arrays are ideal for matrix-vector and matrix-matrix multiplications - the core kernels in many signal and image processing applications. A 2-D mesh connected array is a

natural choice for low-level image processing applications involving data-parallelism mode of computation. However, it has been found that many image processing algorithms can efficiently be mapped onto a 1-D array as well, as demonstrated in the Warp Computer [4]. This has also been borne out by a preliminary study in [6] and a recent simulation study on a similar DSP-based 1-D array architecture [7]. Moreover, compared to a 2-D array, a 1-D array is more efficient from the point of view of array utilization as defined in [8] when solving a host of matrix problems commonly encountered in signal processing. Furthermore, the interconnection between PEs are less complex, and the I/O bandwidth requirement between the host and the array is lower in a 1-D array as compared to a higher dimensional array. Based on these arguments, we have opted for a 1-D array architecture.

2.3.2 Processing Element Architecture :

The ability of a fixed topology array to efficiently map a wide variety of problems depends on :

- * the degree of programmability and computational power of the PE, and
- * the various communication patterns the array architecture can efficiently support.

The most important computational models in linear array are the *local* (or *data-parallel*) and *pipelined* (or *systolic*) computational models [9]. Local and point-wise operations, and matrix multiplication are examples of the former while 1-D convolution is an example of the latter. In the local model,

computation of a result takes place entirely within a PE requiring infrequent or no exchange of partial results with neighbours (coarse-grain parallelism). On the other hand, in pipelined model, computation of single result is distributed among all the PEs. The efficient implementation of the local model requires a PE to have multiple high performance functional units operating in parallel with large high-bandwidth memory units for rapidly supplying operands to computational units and storing results back in the memory. Contrary to this, a very high interprocessor communication bandwidth is required for obtaining efficiency in pipelined computation. Further, the bidirectionality of the flow of partial results is also required in certain problems.

It may be concluded that for maximum efficiency, the PE architecture must be capable of concurrently performing two distinct and independent tasks :

- * high speed computation, and
- * high speed communication with its neighbours.

Traditionally the processor arrays have been built using microcoded custom VLSI chips as PE. These chips possess varying degrees of programmability, and computational and communicational capabilities. The Warp Computer is such a system [4] as it employs horizontally microcoded PEs built with commercially available high performance DSP building blocks that are interconnected by multiple data paths. Although, such bit-slice PE architectures are able to deliver high throughput efficiency and could be adopted to give multiple options for interprocessor communication, but they are difficult to program, complex in hardware and thus expensive.

When we are on the look out for an economic solution, DSP microprocessors rise to the occasion. The possibility of using such building blocks (e.g. TMS 320XX, DSP chips, NEC μ PD 7281, Inmos Transputer etc.) has been mentioned in [10]. A number of image processing architectures have been reported based around the transputer chips [11], which uses high speed on-chip serial communication links. Recently Intel has announced an advanced and versatile building block for processor array design called iWarp [12], but unfortunately its availability is restricted.

The DSP chips that are currently available, exhibit high computational power and that too at reasonable cost. But their serious limitation is the non-availability of on-chip parallel I/O ports which are necessary for interprocessor communication. The only way we are left with is to provide memory-mapped I/O ports as communication links. The processor used in our architecture is Analog Devices ADSP 2100 DSP chip - a 16-bit fixed-point device capable of achieving a maximum throughput of 12.5 MOPs at 80 nsec instruction cycle. Although a 32-bit floating-point device like ADSP-21020 (50 MFLOPS sustained throughput) would have been a better choice, we opted for the simpler fixed point device to build a demonstration system to prove our concept. As the two devices are architecturally similar, the exercise with ADSP 21020 could be taken up as the future work. The salient features of this processor are outlined in the next chapter, where we have discussed the overall array architecture.

CHAPTER 3

OVERVIEW OF PROPOSED ARCHITECTURE

In this chapter, we define the array architecture and consider the various modes in which it operates. We start with the salient features of ADSP 2100 - the processor used in the architecture.

3.1 Architecture of ADSP 2100 :

The block diagram of ADSP 2100 system is given in Fig. 3.1, while its internal architecture is illustrated in Fig. 3.2. ADSP 2100 has separate data buses for connecting (off-chip) data and program/data memories (DM and PM respectively). The device contains three independent and parallel computational units, namely ALU, Multiplier (MAC) and Barrel Shifter. The on-chip instruction-cache effectively provides efficient 3-bus operations when executing short program loops. For instance, in a single cycle, it can fetch two operands from off-chip memories and an instruction from the cache. Moreover, such data moves can be done in parallel with an ALU/MAC operation. This fulfills a very important requirement for high speed DSP calculations. Other additional features are its extended dynamic range for multiple hardware circular buffers, and zero-overhead looping and

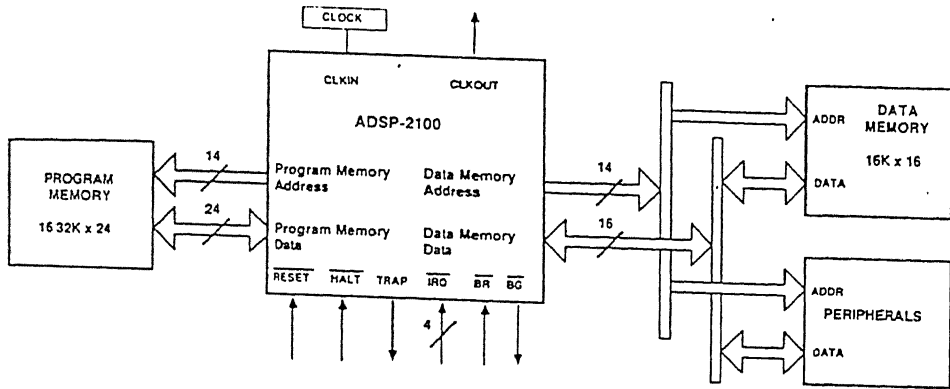


Fig. 3.1 Block Diagram of ADSP 2100 System.

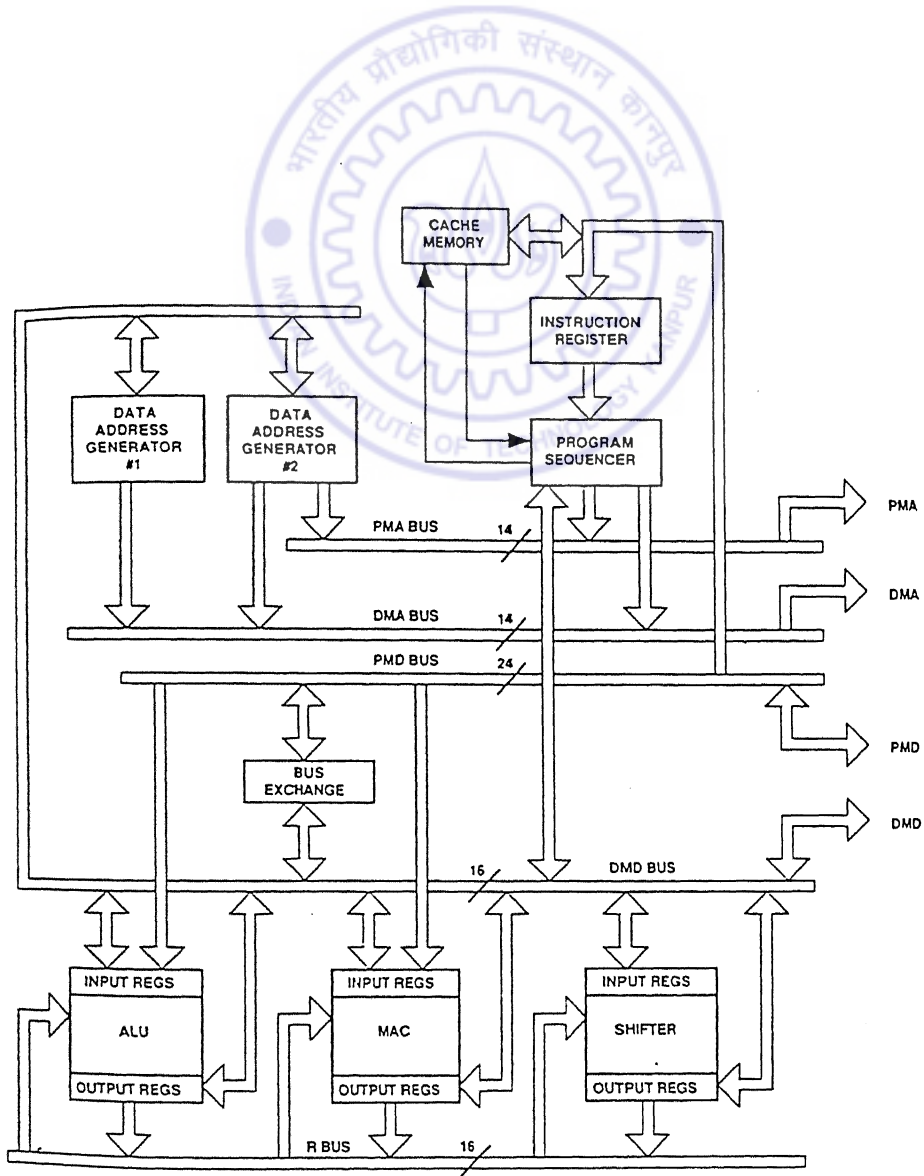


Fig. 3.2 Internal Architecture of ADSP 2100.

branching. The summary of ADSP 2100 key features is given as follows :

- * Separate Program and Data Buses, Extended Off-Chip
- * Single-Cycle Direct Access to 16K x 16 of Data Memory
- * Dual Purpose Program Memory for Both Instruction and Data Storage
- * Single Cycle Direct Access to 16K x 24 (Expandable to 32K) of Program Memory
- * Three Independent Computational Units :
 - Arithmetic/Logic Unit (ALU)
 - Multiplier/Accumulator (MAC)
 - Barrel Shifter
- * Two Independent Data Address Generators
- * Powerful Program Sequencer
- * Internal Instruction Cache
- * Provisions for Multiprecision Computation and Saturation Logic
- * Multifunction Instructions
- * Four External Interrupts

All these facilities together with the cost consideration project ADSP 2100 as a good choice for our architecture.

3.2 Interprocessor Communication Links :

Although ADSP 2100 has good computational features, its serious limitation from the point of view of interprocessor communication is that it does not have on-chip parallel ports. As mentioned earlier, the only way to overcome this limitation is through memory-mapped ports. Application experience in

programmable linear arrays (e.g. Warp Computer) has demonstrated that a large class of signal and image processing algorithms can be mapped onto a 1-D array having two communication channels (henceforth referred to as "X-channel" and "Y-channel"). Data to be processed is transmitted by the host over the X-channel, while Y-channel is used for carrying intermediate/final results.

3.3 Proposed Architecture :

Figure 3.3 shows the block diagram of the proposed architecture. The array is interfaced to the host (PC-AT, in our case). Each PE consists of an ADSP 2100 processor, two memories (PM and DM) and two memory-mapped FIFOs - XFIFO and YFIFO (in short XF and YF) as communication ports. We have used FIFOs instead of latches in order to relax the tight coupling between PEs. Use of latches would require that the rate of data supplied by a PE (to a port) must match the rate of data consumption (from this port) by its neighbouring PE. This is all right for purely systolic mode of computation, but is clearly restrictive in the more general situation where data generation and consumption rates at two neighbouring PEs could be different. Run time flow control is provided to block a PE trying to overwrite a full FIFO, or when it tries to read an empty FIFO. Besides the X- and the Y-channels, there is a Broadcast channel as well. Typically, the various channels are utilized as follows:

- * Global channel broadcasts data from the host to the Program Memory (PM) and Data Memory (DM) of all the PEs. This channel can thus be used for initial loading of constants and program op-codes

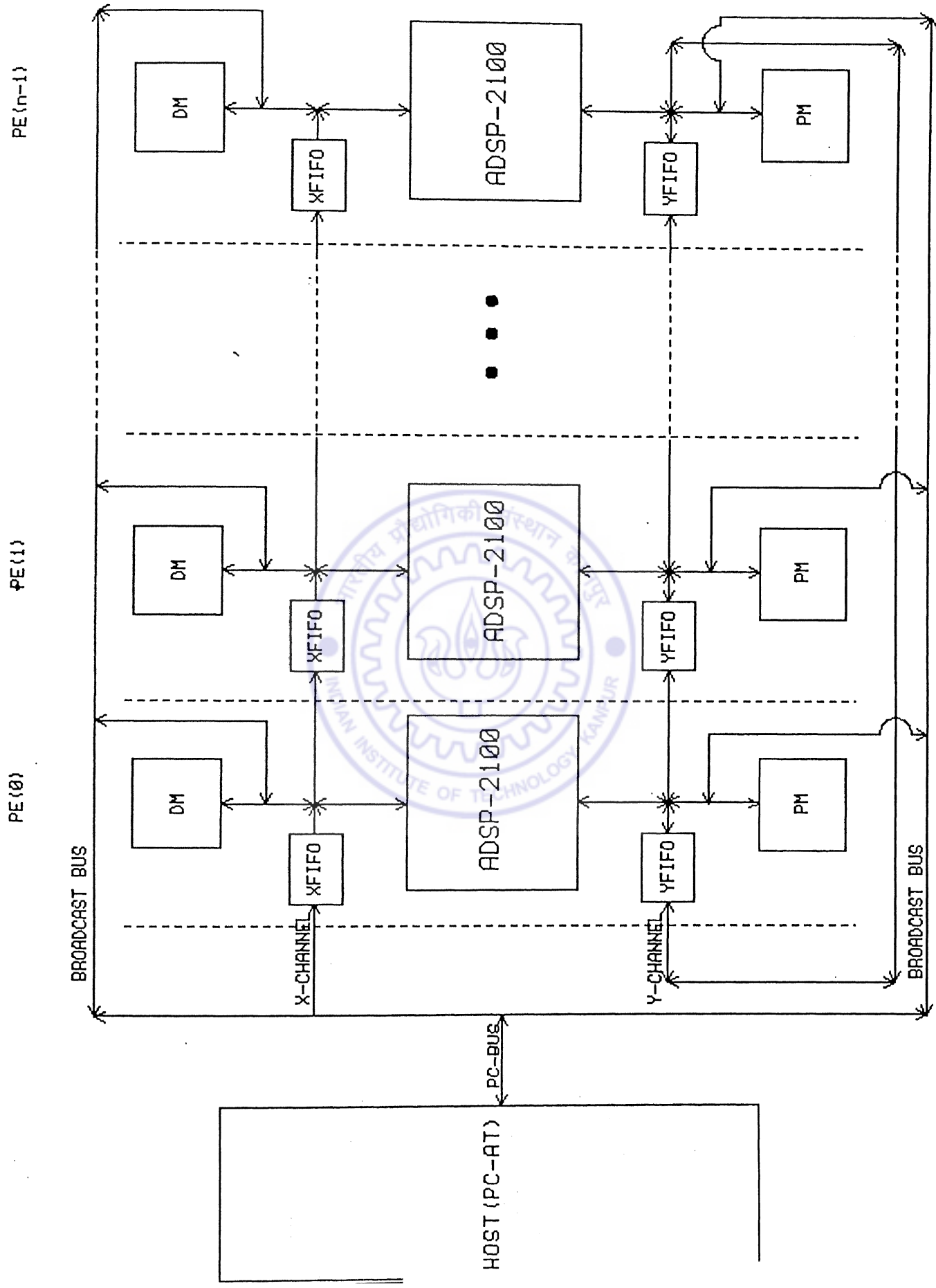


FIG. 3.3 BLOCK DIAGRAM OF PROPOSED ARCHITECTURE.

into the DMs and PMs of the various PEs. It can also be utilized for reading back the results from the memories of PEs into the host. When this channel is being used, I/O and computation can not be overlapped.

* X-channel is a unidirectional one which keeps on getting the raw data from the host and passes it successively to the subsequent PEs. This channel lies on the DM-side of the processor address space. The primary utilization of this channel is thus for systolic communication.

* Y-channel is a bidirectional one meant for communicating the partial results between the adjacent PEs. This channel is not directly accessible to the host. It also forms a ring structure with the output of the last PE being fed to the input of the first PE. This facilitates the recycling of partial results through the array the desired number of times. This feature is very useful in implementing large-size problem on a small size physical array. Bidirectional feature in Y-channel is useful in algorithms like AR-filtering. The ports on this channel are mapped into the PM-side of processor's address space.

3.3.1 Types of Data Transfers Supported :

Because one pair of ports X_{F1} and X_{F1+1} (or Y_{F1} and Y_{F1+1}) shares the same DMD (or PMD) bus, an input to the processor and an output from it can not be done in the same cycle. This implies that highest efficiency (approaching 100 %) can not be achieved in those (systolic) algorithms which require two data items X_{in} and Y_{in} to be input and two others X_{out} and Y_{out} to be output every

cycle (e.g. 1-D convolution). There are other algorithms where such data transfer requirements arise. Let us consider the case of multiplication of two matrices A and B. One possible parallel implementation of this problem is to store equal number of columns (number of columns has to be a multiple of the number of PEs) of matrix B in the local memories of the PEs and then supply each row of A to all the PEs via X-channel. When a row enters a PE, not only it is used to compute the dot product with one resident column, but it must also be stored in the local memory for computing dot products with the other resident columns. Further, it must also be passed-on to the next PE. ADSP 2100 requires three processor cycles to accomplish these tasks :

XFi to reg[2100] ; get element a_{kl} from input FIFO to CPU register.

reg[2100] to DM[2100] ; store a_{kl} in data memory.

reg[2100] to XF_{i+1} ; send a_{kl} to input FIFO of next PE.

Communication rate in such a case becomes just $\frac{1}{3}$ rd of computation rate. This is clearly undesirable in systolic mode of computation. The problem can be circumvented by using external hardware such that when X-data is read into a PE, (if desired) it is simultaneously strobed into the output FIFO as well as to the memory. In addition to the normal data transfers, we have therefore, provided (by incorporating extra hardware [13] in our design) the following types of single-cycle data transfers for efficient systolic computation :

X-channel

- * Input port to processor CPU (normal).

- * Input port to CPU + output port.
- * Input port to CPU + storage (local memory).
- * Input port to CPU + storage + output port.
- * CPU to output port (normal).
- * CPU to storage and vice-versa (normal).
- * CPU to storage + output port.
- * Storage to CPU + output port.

Y-channel

- * YF_1 to CPU and vice-versa (normal).
- * YF_1 to CPU + storage.
- * CPU to YF_{1+1} and vice-versa (normal).
- * CPU to storage and vice-versa (normal).
- * CPU to storage + YF_{1+1} .
- * CPU to YF_1 + storage.
- * YF_{1+1} to CPU + storage.

To the programmer, the multiple-destination data transfers appear as "hardwired single-cycle MACRO instructions". The design details are given in Chapter 5.

3.3.2 Functioning of the Array :

In systolic mode of computation, the functioning of the array goes as follows. Initially, the data constants and program op-codes are broadcast from the host to the PMs and DMs of all PEs through the global channel. Then each PE takes the raw data from the X-channel, read the constants from either DM or the data part of PM (henceforth referred as PMDM), operate on the two, read the partial result from the previous PE through the Y-channel, update

this partial result with the operated data and then pass on the modified partial result to the next PE through the Y-channel, together with the raw data through the X-channel. This way each PE successively operates on the raw data, update the partial result and then in the end, the final result is stored in either DM or PMDM. The host can then read these results through the global channel. The ring structure of the Y-channel comes to our rescue in the cases where partial result has to be updated on multiple passes before obtaining the final result.

In local mode of computation, results are typically computed and stored locally in respective memories and, at the end of the computation these results are read by the host.

Design and assembly of the overall array is divided into two main modules :

- * Design of the Interface card
- * Design of the Processing Element (PE) card

3.3.3 Interface Card :

This is a card which gets plugged into the I/O slots of the PC-AT host. Connection are taken out of this card through a flat cable to the Back-plane card of the array. At the block diagram level (Fig. 3.4), this card consists of a control register, a status register and a set of buffers. The control register issues control signals like the software reset, bus request, PE select lines, broadcast control, memory select control and halt lines. The status register monitors lines like bus grant, full flag of XFIFO and TRAP signals from the processor array. The set of

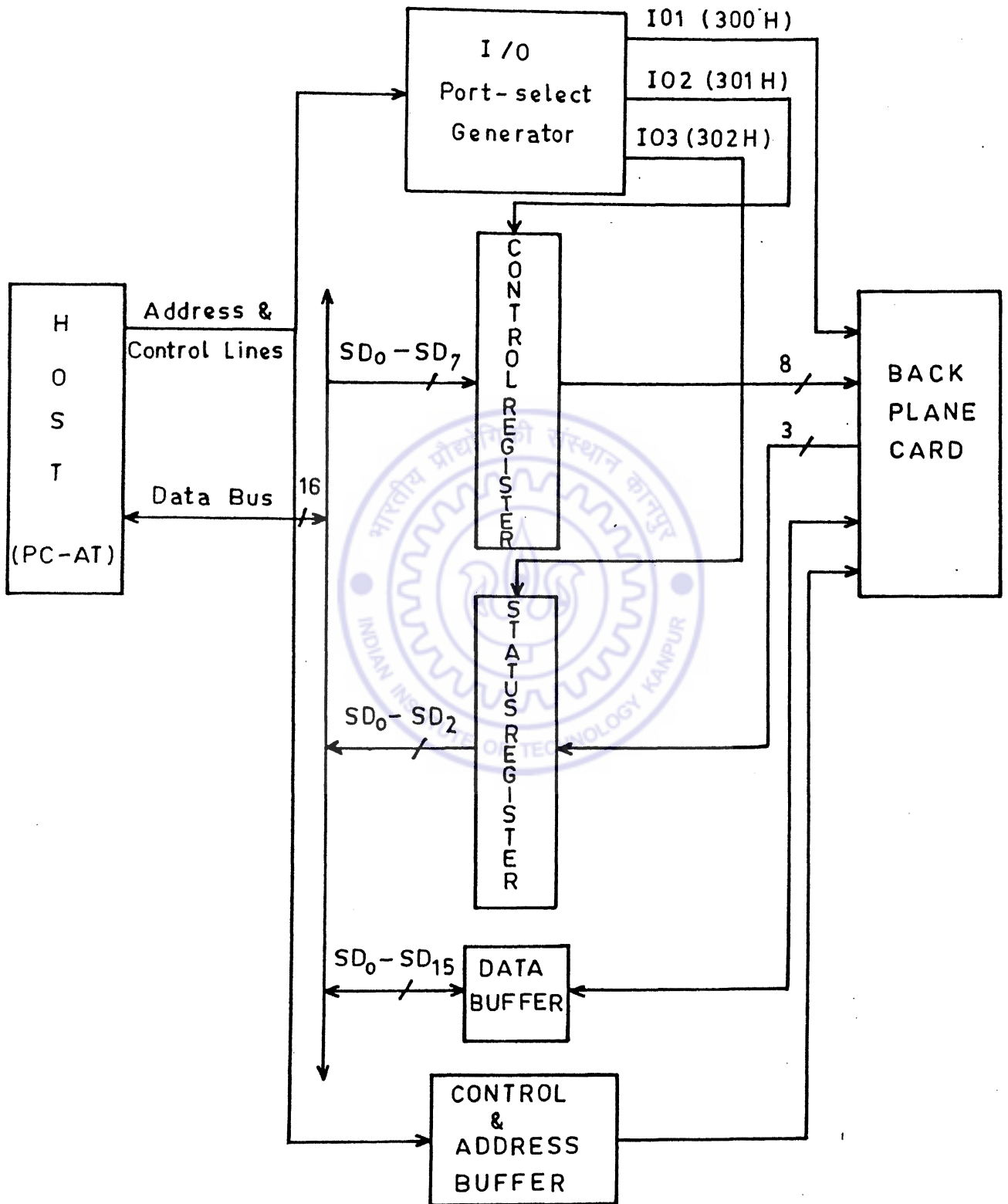


Fig.3.4 Block Diagram of Host Interface.

buffers is employed to buffer the PC - address, data and some control lines before putting them on to the flat cable. The detailed design of this card is given in the next Chapter.

3.3.4 Processing Element Card :

This card consists of the processor (ADSP 2100) together with its Program and Data Memory (PM and DM respectively) and the FIFO ports on X- and Y-channels. Besides, as mentioned earlier, there are two main facilities endorsed in this card. Firstly, extra hardware is provided to enhance the interprocessor communication options in the form of single-cycle multiple-destination data transfer capabilities. Second facility is to make accessible to the host the PMs and DMs of each PE through the global channel, for the purpose of downloading/broadcasting program/data. Once again, the detailed design of PE card is dealt with separately in Chapter 5.

3.4 Modes of Operation of the Processor Array :

There are two main modes of operation of the processor array:

- * Off-line Mode
- * On-line Mode

In the off-line mode, the global channel is in use. This mode is used for the initial downloading of program to the PMs of all the PEs, broadcasting of constants (e.g. filter weights, a matrix etc.) to the DMs and PMDMs, and for reading back the final results from the memories of the PEs to the host. In this mode, I/O and computation cannot be overlapped.

In the on-line mode, the X- channel is supplied with the raw data from the host. The primary use of this mode is for systolic computation. Here, the I/O and computation operations can be overlapped. So it can be utilized for the real-time signal processing of data coming from, say, an A/D converter.



CHAPTER 4

DESIGN OF HOST INTERFACE AND BACKPLANE

The array behaves as a slave -processor to a general purpose host (PC-AT/386, in our case). The host being the master, has to perform the following functions :

- * To broadcast program and data (constants/parameters) to the PMs and DMs of all the PEs.
- * To give a software reset to all the PEs so as to signal them that they may start the useful execution now.
- * To keep on supplying the streams of raw data (the one to be processed) through the X-channel.
- * To read back the results from the memories of the PEs through the broadcast bus.

In order to perform these functions, the host has to generate and issue various control signals to the array, as well as to monitor the various status conditions and initiate appropriate actions accordingly. The host interface basically consists of the I/O port-select signal generator, a control register, a status register and the buffers for buffering the address, data and control lines of the host, before broadcasting them to the processing elements as the global channel. These various units are

described as follows :

4.1 I/O Port-Select Signal Generator :

Function of this generator is to provide chip select signals for control register, status register and XFIFO (of the first PE), which are I/O mapped to host I/O space as follows (in the present system):

XFIFO (of first PE) : Port 300 H (IO1 in Fig 4.1)

Control Register (CR) : Port 301 H (IO2 in Fig 4.1)

Status Register (SR) : Port 302 H (IO3 in Fig 4.1)

In order to give flexibility to the I/O ports- select signal generation, an Erasable Programmable Logic Device (Intel 5C032) is used as the I/O port-select generator. The details of the circuitry implemented in the EPLD is given in Fig 4.1.

4.2 Control Register :

It is an 8-bit output port mapped at I/O address 301 H. A permanently enabled latch (74LS373) is used to implement the control register. The bit assignment for the control signals is as follows :

Bit 0 : $\overline{\text{HRS}}$ (Host Reset)

Bit 1 : $\overline{\text{HBR}}$ (Host Bus Request)

Bit 2 : PESL (PE Select, Lower bit)

Bit 3 : PESH (PE Select, Middle bit)

Bit 4 : PESU (PE Select, Upper bit)

Bit 5 : $\overline{\text{BC}}$ (Broadcast Control)

Bit 6 : $\overline{\text{HLT}}$ (Halt)

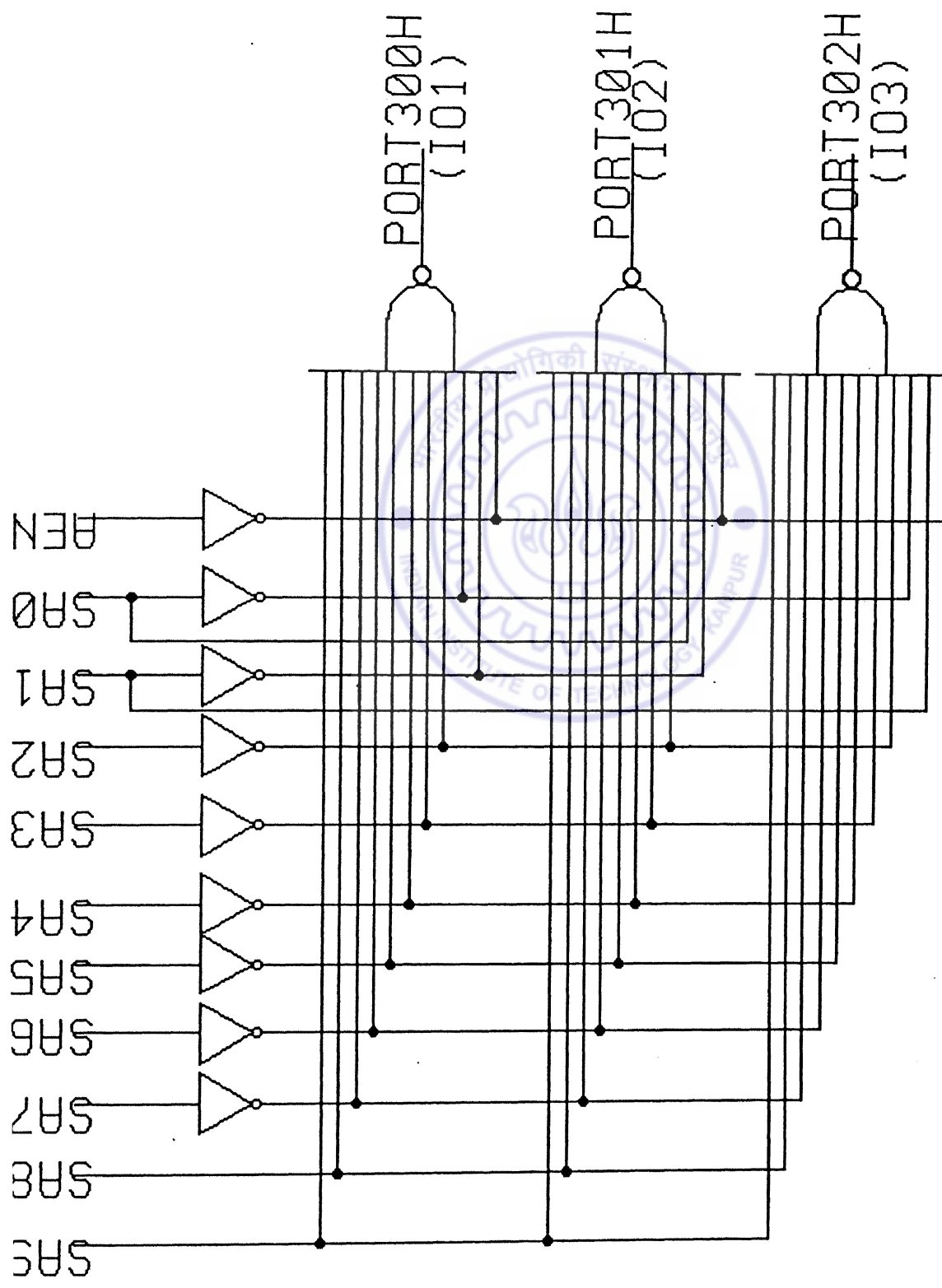


FIG 4.1 I/O PORT-SELECT GENERATOR.
(EPLDIO-5C032)

Bit 7 : MS (Memory Select control)

$\overline{\text{HRS}}$: This signal is used by the host to issue software reset to all the PEs in the array. It resets all the processor (ADSP 2100) and all the FIFOs.

$\overline{\text{HBR}}$: Through this signal, the host requests the control of the PE memory buses for directly accessing the array memories.

PESL, PESM, PESU :

These signals are used to select a particular PE. A maximum of 8 PEs can thus be addressed.

$\overline{\text{BC}}$: It is used for downloading the same program and/or data to all the PEs. Whenever $\overline{\text{BC}}$ is asserted, the PES lines become ineffective.

$\overline{\text{HLT}}$: The assertion of this bit by the host issues $\overline{\text{HALT}}$ signal to all the processors (ADSP 2100).

MS : By asserting MS=0, the host can access the upper and middle byte of PM. And by asserting MS=1, the host can access the lower byte of PM as well as DM and PMDM. The use of this signal is further elaborated in sec 5.

4.3 Status Register :

It is an 8-bit input port mapped at I/O address 302 H. A 74LS245 buffer is used to implement this register. The buffer is enabled by the $\overline{\text{IOR}}$ signal of the host. The bit assignment of the status signals is as follows :

Bit 0 : $\overline{\text{BGH}}$ (Bus Grant to Host)

Bit 1 : TRAPH (Trap signal from array to the Host)

- Bit 2 : $\overline{\text{FFH}}$ (Full Flag signal from XFIFO of PE1 to the Host)
- $\overline{\text{BGH}}$: This signal is obtained by successively ORing the Bus Grant ($\overline{\text{BG}}$) signals from individual PEs. This implies that only when all the PEs have released their buses, then $\overline{\text{BGH}}=0$ and host can now access the array memories.
- TRAPH : This signal is obtained by successively ANDing the TRAP signals from individual PEs. ADSP 2100 issues TRAP signal whenever it executes the TRAP instruction. This facility is used to indicate the end of the program by the PEs to the host.
- $\overline{\text{FFH}}$: This signal is generated whenever XFIFO of the first PE is full. This status signal may be checked by the host before each data transfer to XFIFO of PE1 (polled data transfer). This signal may also be used to interrupt the host, so that the host may go into the wait state till the XFIFO is no more full.

4.4 Loading of XFIFO :

XFIFO of the first PE is configured as the 16-bit output port mapped at I/O address 300 H of the host. The data transfer from the host to the XFIFO being 16-bit wide, the $\overline{\text{IOCS16}}$ signal to the host must be asserted. This is accomplished by buffering the XFIFO-select signal (IO1) from the EPLD and feeding it to $\overline{\text{IOCS16}}$ pin of the host.

4.5 Data, Address and Control Buffers :

Four 74LS245 buffers are used for buffering the data lines

(SD₀-SD₁₅) and address lines (SA₀-SA₁₅). A fifth buffer (74LS245) is used to buffer the control lines $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, SBHE and AEN of the host, and the $\overline{\text{WR}}$ signal to XFIFO of PE₁, before being taken out through the flat ribbon cables. The same buffer is also used for asserting $\overline{\text{IOCS16}}$ signal of the host, as mentioned in sec. 4.3.

4.6 Interrupt Circuitry :

As mentioned in sec. 4.3, the signal $\overline{\text{FFH}}$ may be used to interrupt the host. The facility is provided that this $\overline{\text{FFH}}$ is fed (if desired) through a jumper to anyone interrupt (whichever is free) of the host. The various host interrupts that may be chosen through the jumper are IRQ₅, IRQ₁₀, IRQ₁₁, IRQ₁₂, IRQ₁₅. The $\overline{\text{FFH}}$ signal is latched (into 74LS374) at the rising edge of each I/O write ($\overline{\text{IOW}}$) of host, and then at the next I/O write, it is fed to the selected interrupt (through the jumper), so that whenever $\overline{\text{FFH}}$ gets asserted, the next I/O write gets interrupted. That is how the host goes into the wait-state and remains there till the XFIFO of the first PE is again ready to receive data from the host. In this way, whenever the XFIFO of the first PE is full, the data transfer to it from the host may be stopped temporarily either through the status check (as mentioned earlier), or through the interrupt service subroutine of the host.

4.7 Interface Card Assembly :

The interface circuitry is assembled on half size PCB that plugs directly into the I/O extension slot of the host. All the data, address and control lines discussed in the preceding

sections are taken out of this assembly to the back-plane card through two 50-pin FRC connectors. The pin assignment of these connectors is given below :

FRC CONNECTOR 1

Odd-numbered pins :-

Pin 1	: Buffered \overline{WR} signal to XFIFO of PE1
Pin 3	: Buffered \overline{MEMR}
Pin 5	: Buffered \overline{MEMW}
Pin 7	: Buffered SBHE
Pin 9	: \overline{HRS}
Pin 11	: \overline{HBR}
Pin 13 to Pin 17	: PESL, PESH, PESU
Pin 19	: \overline{BC}
Pin 21	: \overline{HLT}
Pin 23	: MS
Pin 25	: \overline{BGH}
Pin 27	: TRAPH
Pin 29	: \overline{FFH}
Pin 31 to Pin 49	: Buffered SDo-SD9

All even numbered pins are grounded so as to make alternate wires at ground and thus avoid cross-talk.

FRC CONNECTOR 2

Odd-numbered pins :-

Pin 1 to Pin 11	: Buffered SD10-SD15
Pin 13 to Pin 43	: Buffered SA0-SA15
Pin 45	: Vcc (+5V)
Pin 47	: $\overline{MEMCS16}$

Pin 49 : Buffered AEN

All even-numbered pins are grounded so as to avoid cross-talk in the signals.

The $\overline{\text{MEMCS16}}$ connected to pin 47 of FRC connector 2 is generated in the array whenever the host wants to download 16-bit data to the array memories. The generation of this signal is discussed in sec. 5.1.

The overall schematic of the interface circuit is given in Appendix A. The chip count for this card is coming out to be 11. Physical placement of the chip is given in Appendix B.

4.8 Design of Backplane Card :

This card mainly consists of female-type Euro-connectors and the buses that run all along to accomplish the various interconnections. Its overall capacity is to get mounted on it a total of 8 PE cards. The 49 signals coming from the host through a flat cable are fed to this backplane by two 50-pin FRC connectors. These signals are then buffered before being fed to the broadcast bus that runs all along this card. This broadcast bus feeds one of the Euroconnectors (given the name $\text{EC}_{x1, x=1, \dots, 8}$) of each PE. The constituents of the broadcast bus (47-bit wide) are the 16-bit data lines and 16-bit address lines, both from the host, the control signals from the host $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, SBHE, AEN, and the eight control lines from the control register of the interfacing card, the power and the ground lines and a line which carries the master clock signal. The provision of this master clock in the backplane card is in addition to the clocks that are provided individually

on each PE card. So there is a flexibility of using either the independent clock of each PE or synchronizing all PEs by the master clock. These two clocks are ORed on each PE. So either of them could be used and the other terminal is to be grounded. The software reset coming from the control register of the interface card is ANDed with the Push-button/Power-on resets before being fed to the Broadcast bus. A simple RC circuit together with a D-Flip Flop is used to generate the Push-button/Power-on resets. The circuitry is given in Fig 4.2. The D-Flip Flop is clocked by the Master clock.

Besides these broadcast line, the Euro-connectors EC_{x1} is also fed with the inputs to the XFIFOs.. The output of each XFIFO is to be fed to the input of the XFIFO of the next PE. In the first PE, the XFIFO inputs (X_{in}) are fed directly by the host through its data lines (SD₀ to SD₁₅), and XFIFO outputs (X_{out}) of the last PE are connected nowhere. The XFIFO outputs (X_{out}) , the YFIFO inputs (Y_{in}) and outputs (Y_{out}), the control lines inputs (C_{in}) and outputs (C_{out}) are all being accomplished in another Euroconnector given the name EC_{x2}, (x=1,...,8). The schematic layout of these Euroconnectors is given in Fig. 4.3.

The data bus of the X-channel is terminated after the last PE, whereas the data bus of the Y-channel together with the control signals is fed back from the last PE to the first PE so as to form a ring structure. This facilitates the rotation of partial results the desired number of times before obtaining the final result. There may often arise a situation where lesser than 8 PEs are mounted on the backplane. In such a situation, therefore, it

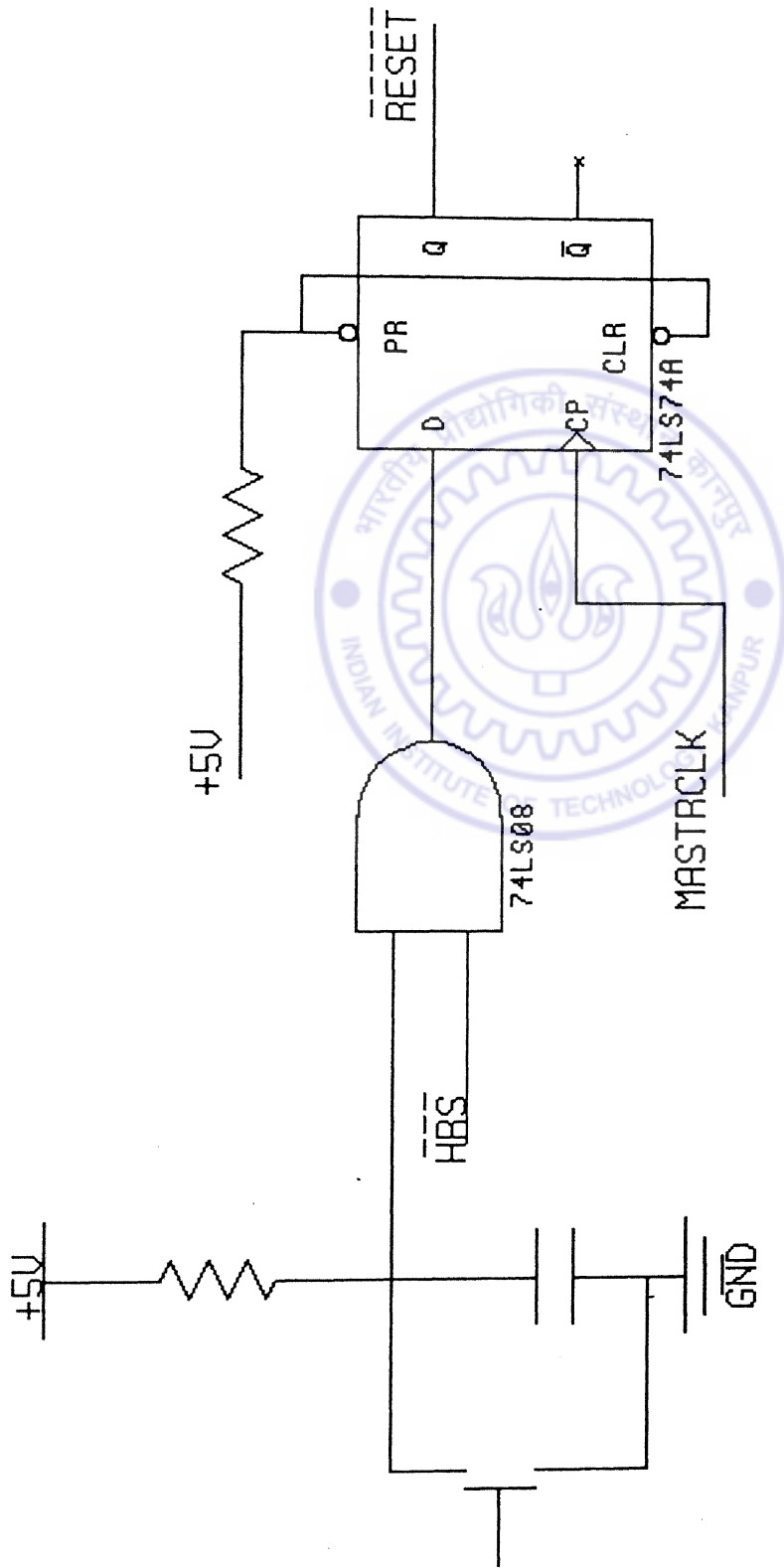


FIG 4.2 RESET CIRCUITRY

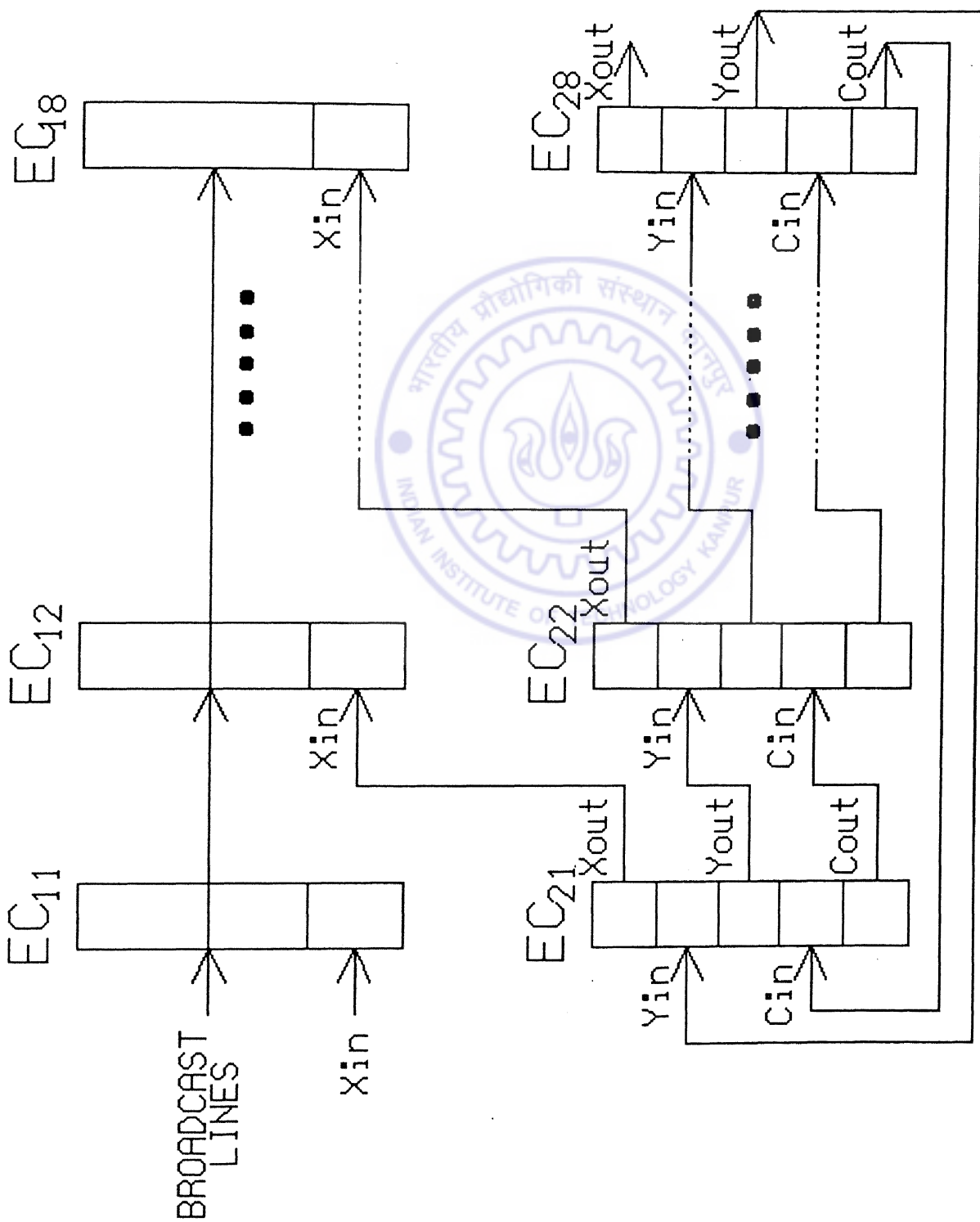


FIG 4.3 LAYOUT OF BACKPLANE

becomes necessary to provide connectors to feed the signals back from the last PE (whatsoever) to the first PE. This is done using a flat cable, and 50 pin FRC connectors are used for this purpose.

4.9 Sequence of Operations :

When a processing work has to be carried out by the array processor, the following sequence of operations is typically adopted :

Firstly, the host writes the control word 61H into the control register so as to assert the Bus Request (\overline{BR}) to all the PEs. It then waits and keeps on checking the status by reading the status register till all the PEs have given their Bus Grants (\overline{BG}). After that the host writes MS=0 and puts appropriate values on PESS and \overline{BC} lines through the control register. The upper and middle bytes of PM of all the PEs are then loaded. The MS is then changed to zero and the lower bytes of PM, and DM and PMDM are uploaded. Once the uploading is over, the host writes 62H into the control register and resets all the PEs. The reset (\overline{HRS}) is removed after sufficient time by writing 63H into the control register. The program execution starts in the PEs. Simultaneously the host starts supplying the raw data into the X-channel by writing into its port IO1. The program in each PE ends with a TRAP instruction, so that their TRAP lines are set when they have finished their program execution. The host by checking its status register gets to know that all the processors have given the TRAP and that the results are ready to be ready to be read back from the PEs. It asserts \overline{BR} once again, waits for \overline{BC} of all PEs, puts

appropriate addressing values in its control register and read the results from the local memories (PMDM and/or DM) of each PE. Finally the host writes 23H into its control register so as to issue HALT signal to all the processors.



CHAPTER 5

DESIGN OF PROCESSING ELEMENT

The PE card basically consists of the ADSP 2100 processor, its Program Memory (PM), Data Memory (DM), their interface to the global channel from the host, the FIFOs on the X-channel and Y-channel, and the extra hardware to facilitate single-cycle multiple-destination data transfer options. Fig 5.1 shows the main parts of a PE. A brief description follows :

- * DMT stands for the Data Memory used for accomplishing the single-cycle multiple-destination data transfers on X-channel. Details will be given in sec 5.2.
- * DM stands for the memory local to the processor on the DM-side. This memory is accessible to host as well.
- * PMT stands for the Program Memory used for facilitating the single-cycle multiple-destination data transfers on the Y-channel. Details will once again be given in sec 5.2. It is to be noted that it is a portion of the data part of the program memory.
- * PMDM is the memory local to the processor on the PM-side. This memory is also accessible to host. It is also a portion of the data part of the program memory.
- * PM is the program part of the Program Memory.
- * XF1 is the input FIFO on the X-channel.
- * YF1 is the forward input FIFO on the Y-channel.

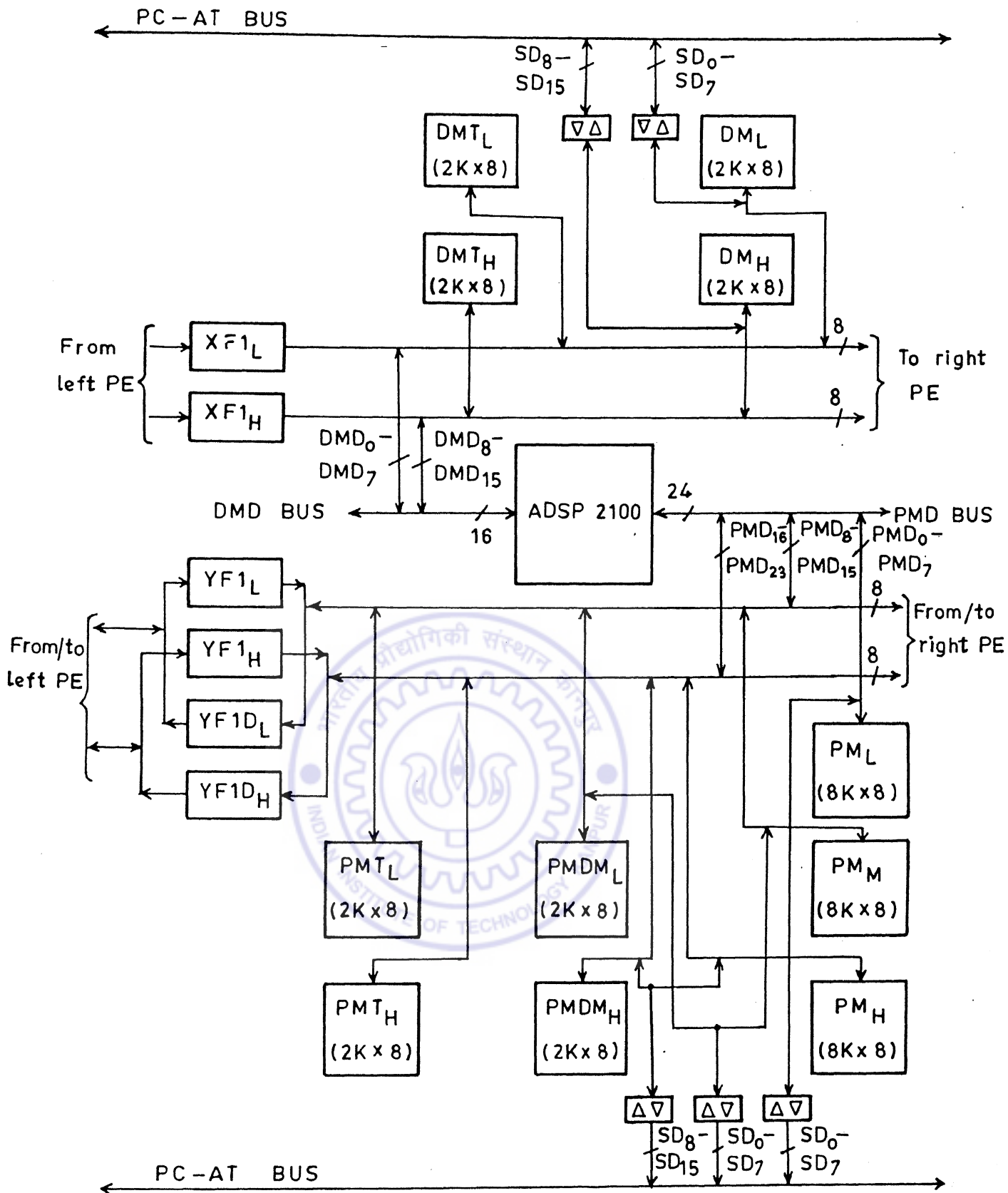


Fig. 5.1 Block Diagram of Processing Element .

* YF1D is the backward output FIFO on the Y-channel.

Design of PE card can be divided into following three parts :

- . Design of interface to host global channel
- . Design of single-cycle multiple-destination data transfer circuit
- . Run-time flow control

5.1 Design of Interface to Global Channel :

As discussed earlier, host uses the global channel for downloading of program and data as well as reading back results. This requires that PM and DM of each PE must be mapped onto the host memory-address space. In our design, we have provided 2 Kilowords (each word of 2 Bytes) of DM and PMDM, and 8 Kwords (each word of 3 Bytes) of PM. This would require a host memory space of $(2K \times 2 B + 2K \times 2 B + 8K \times 3 B) = 32KB$. This requirement can be halved with the help of Memory Select (MS) control bit, which effectively acts like an extra address line. These memories are mapped onto the host memory space as follows :

	SA ₁₅ -SA ₀	Selected Memory
MS = 0	0000 to 3FFF H (16 KB)	PM's higher byte at even addresses and middle byte at odd addresses.
MS = 1	0000 to 1FFF H (8 KB)	Lower byte of PM.
	2000 to 2FFF H (4 KB)	PMDM's higher byte at even addresses and lower byte at odd addresses.
	3000 to 3FFF H (4 KB)	DM's higher byte at even addresses and lower byte at odd addresses.
Total host memory space used is 16 KB.		

The various local memories are connected to the global bus through bidirectional buffers (74LS245). The chip-selects from the host side for these memories is generated by an EPLD (Intel 5C032) called EPLDCS. The same signals are used for enabling the corresponding buffers also. The circuit implemented inside EPLDCS is shown in Fig 5.2. From the processor side, the local memory DM is mapped in 0 to (2K-1) slot of DM-address space, while the other local memory PMDM occupies 0 to (2K-1) slot of data part of PM-address space. The Program Memory PM is lying in 0 to (8K-1) slot of the program part of the PM-address space.

Referring to Fig 5.3, whenever the host has to communicate with any of the PEs, it activates the $\overline{\text{HBR}}$ control, which is fed to Bus Request ($\overline{\text{BR}}$) terminal of the ADSP 2100 of each PE. The consequent $\overline{\text{BG}}$ from the processor is Ored with the $\overline{\text{BG}}$ coming from the right PE (called $\overline{\text{BGHI}}$) to generate $\overline{\text{BGHO}}$ for the left PE. The host keeps waiting till it receives the $\overline{\text{BGHO}}$ from the leftmost PE (i.e. all the PEs have given the Bus Grant). After that the host selects a particular PE by asserting appropriate values on PE Select lines PESL, PESM, PESU. In case the same data is to be broadcasted to all PEs, the host activates the Broadcast Control ($\overline{\text{BC}}$). The output of PES/ $\overline{\text{BC}}$ control logic is Ored with $\overline{\text{BG}}$ to generate the signal U1, which in turn, is used for enabling the chip selects for the various local memories.

In our design, the data/program broadcasting to the DM, PMDM and the higher and middle bytes' chunk of PM is word-wise (each word of 16-bits), while it is byte-wise for the lower byte chunk of PM. It, therefore, necessitates the assertion of $\overline{\text{MEMCS16}}$ signal

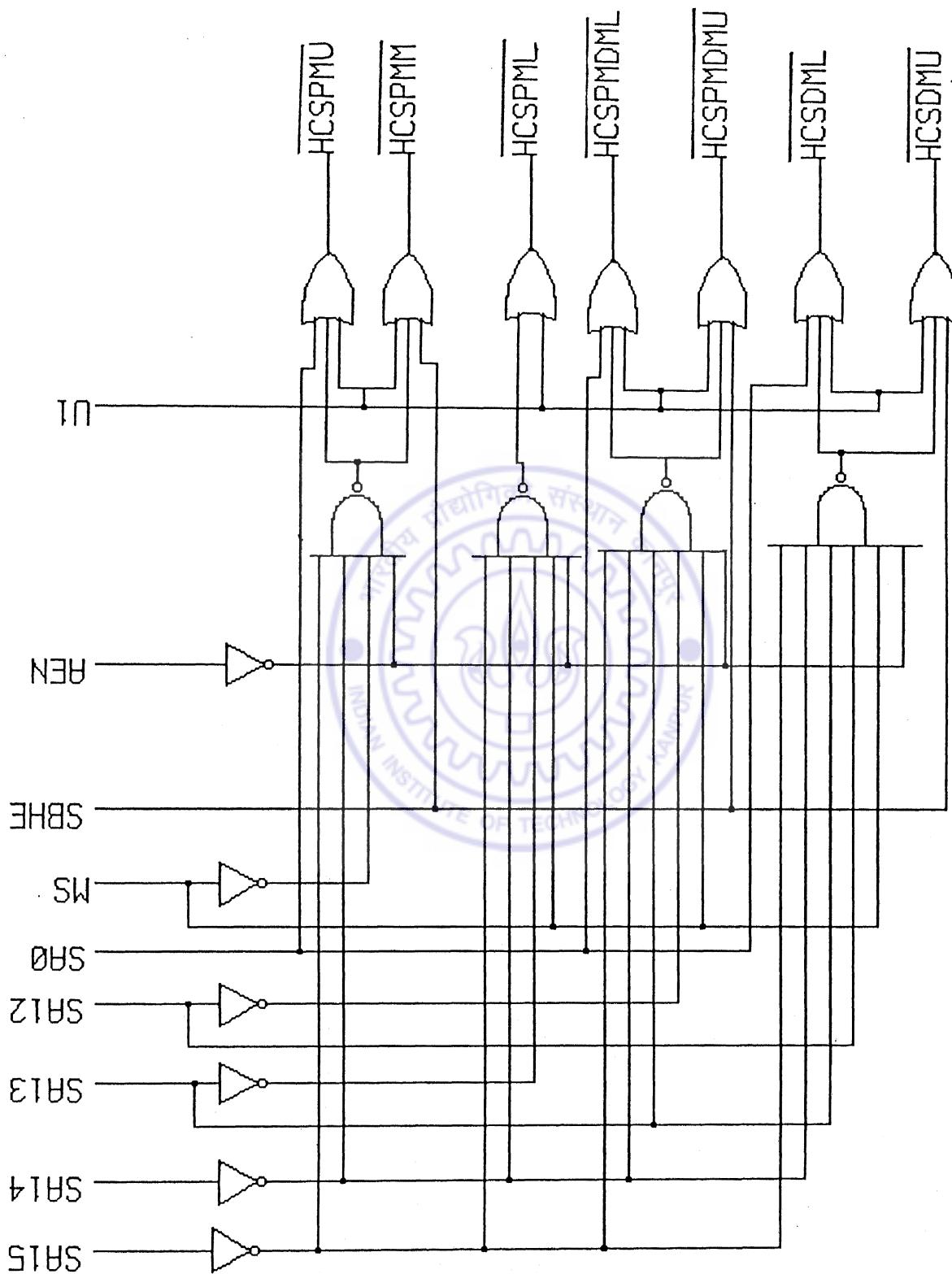


FIG. 5.2 EPLD FOR MEMORY CHIP-
SELECT GENERATION

to the host, whenever any of the DM, PMDM or the higher-middle bytes' memory chunk of PM is selected by the host. This is done by ANDing the chip selects of all the memories requiring word-wise data transfers, as well as $\overline{\text{MEMCS16}}$ from the right processors (called $\overline{\text{MEMCS16I}}$). For schematic, refer to Appendix C.

5.2 Design of Single-Cycle Multiple-Destination Data Transfers Circuits:

Table 5.1 shows the various single-cycle-multiple-destination (SCMD) data transfers on the X-channel. A 2 Kiloword memory (DMT) together with a 3 to 8 decoder and a few gates is used to accomplish these data transfers. Whenever the address on DMA lines is between 2K to (4K-1) and a Data Memory Write operation is carried out (i.e. $\overline{\text{DMWR}}$ is active), DMA_{13} to DMA_{11} are decoded and fed to the gates in such a way that the data gets transferred from the processor (more precisely, the processor's register that is mentioned in the instruction) to the FIFO of the next PE (i.e., XF2). Similarly, when the address on DMA line is between 6K to (8K-1), the data gets transferred from DMT to the processor as well as XF2 under $\overline{\text{DMRD}}$ control, and from processor to DMT as well as XF2 under $\overline{\text{DMWR}}$ control. While the DM-address space between 14K to (16K-1) is used for data transfer from XF1 to processor, DMT as well as XF2 under $\overline{\text{DMRD}}$ control, the same DM-slot is used to activate the Retransmit control of the XFIFO under the $\overline{\text{DMWR}}$ control. There is no operation being carried out in the DM-slot of 2K to (4K-1) under $\overline{\text{DMRD}}$ control, and 8K to (14K-1) under $\overline{\text{DMWR}}$ control. On similar lines, rest of the data transfers could be

DMA ₁₃	DMA ₁₂	DMA ₁₁	Control : $\overline{\text{DMRD}}$	Control : $\overline{\text{DMWR}}$	DM-Slot
0	0	0	DM to Pr (local)	Pr to DM (local)	0 to (2K-1)
0	0	1	NOP	Pr to XF2	2K to (4K-1)
0	1	0	DMT to Pr	Pr to DMT	4K to (6K-1)
0	1	1	DMT to Pr & XF2	Pr to DMT & XF2	6K to (8K-1)
1	0	0	XF1 to Pr	NOP	8K to (10K-1)
1	0	1	XF1 to Pr & XF2	NOP	10K to (12K-1)
1	1	0	XF1 to Pr & DMT	NOP	12K to (14K-1)
1	1	1	XF1 to Pr & DMT & XF2	Retransmit control of XFIFOs	14K to (16K-1)

Table 5.1 Single-Cycle Multiple-Destination Data Transfers on X-Channel.

PMA ₁₃	PMA ₁₂	PMA ₁₁	Control : $\overline{\text{PMRD}}$	Control : $\overline{\text{PMWR}}$	PM-Slot
0	0	0	PMDM to Pr (local)	Pr to PMDM (local)	16K to (18K-1)
0	0	1	YF2D to Pr	Pr to YF2	18K to (20K-1)
0	1	0	PMT to Pr	Pr to PMT	20K to (22K-1)
0	1	1	YF2D to Pr & PMT	Pr to PMT & YF2	22K to (24K-1)
1	0	0	YF1 to Pr	Pr to YF1D	24K to (26K-1)
1	0	1	NOP	NOP	26K to (28K-1)
1	1	0	YF1 to Pr & PMT	Pr to YF1D & PMT	28K to (30K-1)
1	1	1	INTBUF	NOP	30K to (32K-1)

Table 5.2 Single-Cycle Multiple-Destination Data Transfers on Y-Channel

understood from this table. Whenever the data transfer involves DMT, the address to DMT is issued by the lower 11 DMA lines, namely DMA₀ to DMA₁₀. The 0 to (2K-1) slot of DM-address space is used by the processor to communicate with its local Data Memory (given the name as DM). The combinational circuit facilitating these data transfers is implemented in an EPLD (Intel 5C032) called EPLDX. The circuit implemented in the EPLD is depicted in Fig 5.4.

The design of Y-channel is very much similar to the X-channel with the only difference that Y-channel is bidirectional as against the unidirectional nature of the X-channel. Thus two sets of FIFOs in directions opposite to each other are employed in the Y-channel. The various data transfers permissible in the Y-channel are given in Table 5.2. Here PMT is the 2 Kiloword memory which together with a 3 to 8 decoder and digital gates facilitates these data transfers in the Y-channel in a single cycle. For 2K to (4K-1) slot of PM with PMDA being high, the data transfer takes place from YF2D (i.e. backward YFIFO of the next PE) to the processor under $\overline{\text{PMRD}}$ control and the processor to YF2 (i.e. forward YFIFO on the next PE) under $\overline{\text{PMWR}}$ control. Similarly, with PMDA being high, the 6K to (8K-1) slot affects the data transfer from YF2D to processor as well as PMT under $\overline{\text{PMRD}}$ control, and processor to PMT as well as YF2 under $\overline{\text{PMWR}}$ control. On the same lines are the rest of the data transfers in the table, taking place. The 14K to (16K-1) slot under $\overline{\text{PMRD}}$ gives a control INTBUF, which is used to enable the buffer employed to read the Empty Flag ($\overline{\text{EF}}$) and Full Flag ($\overline{\text{FF}}$) status of the YFIFOs. The combinational

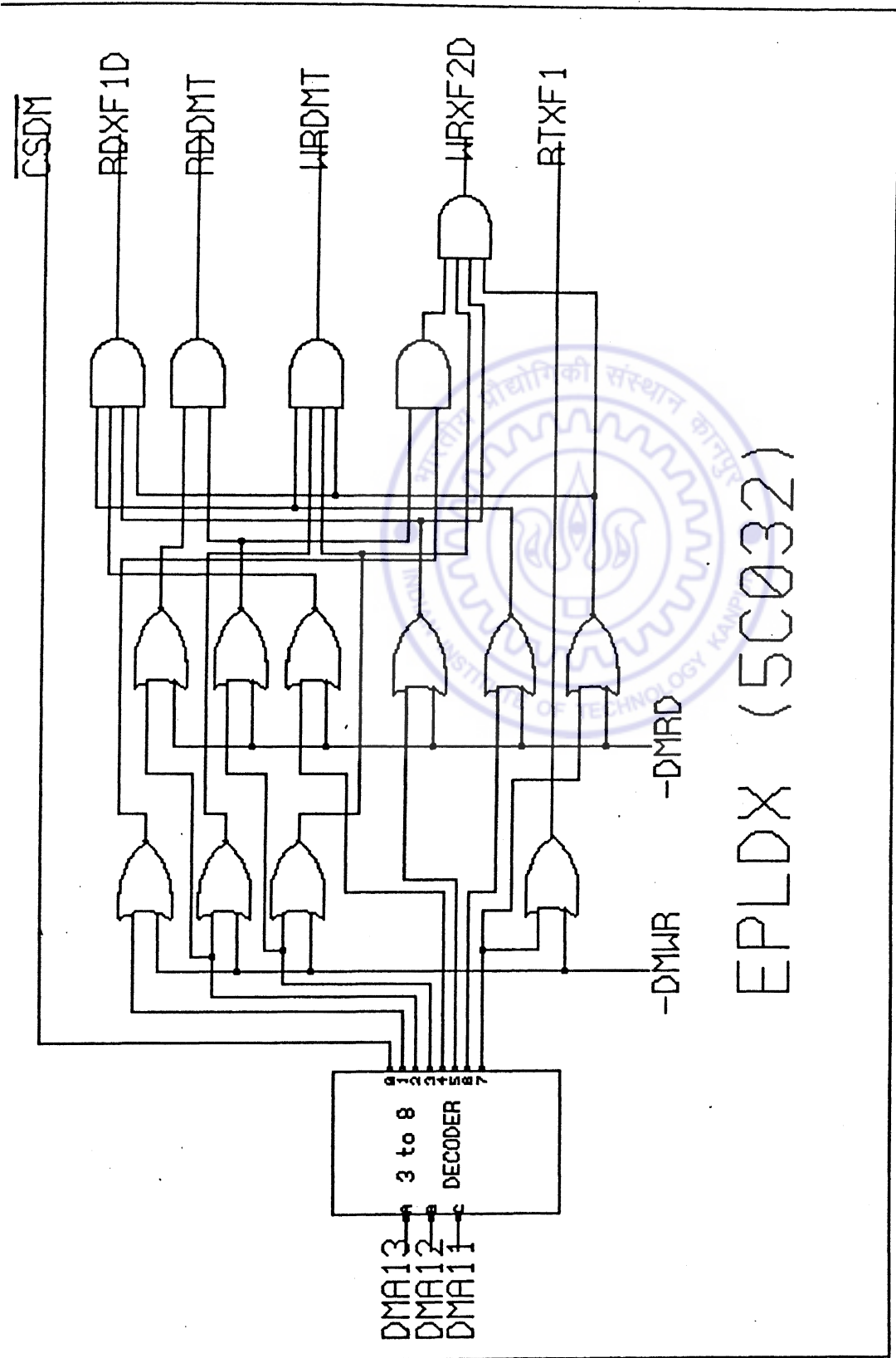


FIG. 5.4 CIRCUIT FOR SCMD DATA TRANSFER IN X-CHANNEL.

circuit facilitating these data transfers in Y-channel is implemented in an EPLD (Intel 5C032) named as EPLDY. The circuit implemented inside this EPLDY is given in Fig 5.5.

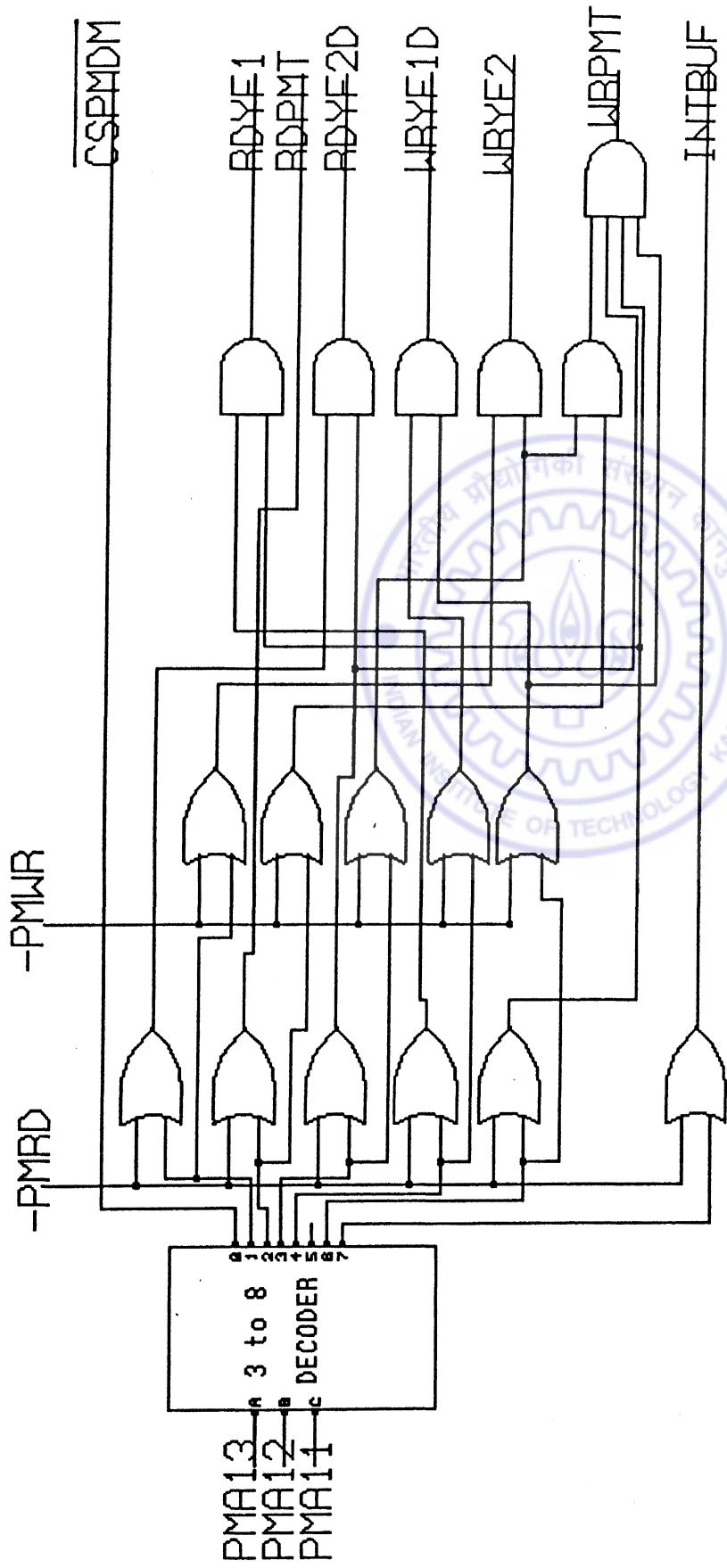
5.3 Run-Time Flow Control :

Whenever a PE tries to read an empty FIFO or tries to write a full FIFO, it must be blocked. In the X-channel, this flow control is implemented in hardware, while in the Y-channel, a software-hardware combination provides the flow control.

5.3.1 X-Channel Flow Control :

In the X-channel, the empty flag of the previous FIFO (i.e. signal $\overline{\text{EFXF1}}$) and the full flag of the next FIFO (i.e. signal $\overline{\text{FFXF2}}$) is monitored before each data transfer by the circuit given in Fig 5.6. The corresponding timing diagram is given in Fig 5.7. The $\overline{\text{EFXF1}}$ and $\overline{\text{FFXF2}}$ signals are latched once in each instruction cycle at the rising edge of the CLKOUT of ADSP 2100. A little after the last valid read from XF1 (or write to XF2), the $\overline{\text{EFXF1}}$ ($\overline{\text{FFXF2}}$) goes low and the same gets latched at the rising edge of CLKOUT. Now, whenever the next data read from XF1 (or write to XF2) is tried, the DMACK signal goes low and ADSP 2100 goes into the wait state and remains like that till fresh data has been written into XF1 by the previous PE (or read from XF2 by the next PE). Soon after the flag signal goes high, the DMACK also goes high and the processor comes out of the wait state, and simultaneously, a read pulse is generated to XF1 (or a write pulse is generated to XF2). This is why the read and write signals being

CENTRAL LIBRARY
I.I.T. DELHI
Doc. No. A117680



EPLDY (5C032)

FIG 5.5 CIRCUIT FOR S.C.M.D.
DATA TRANSFER IN Y-CHANNEL.

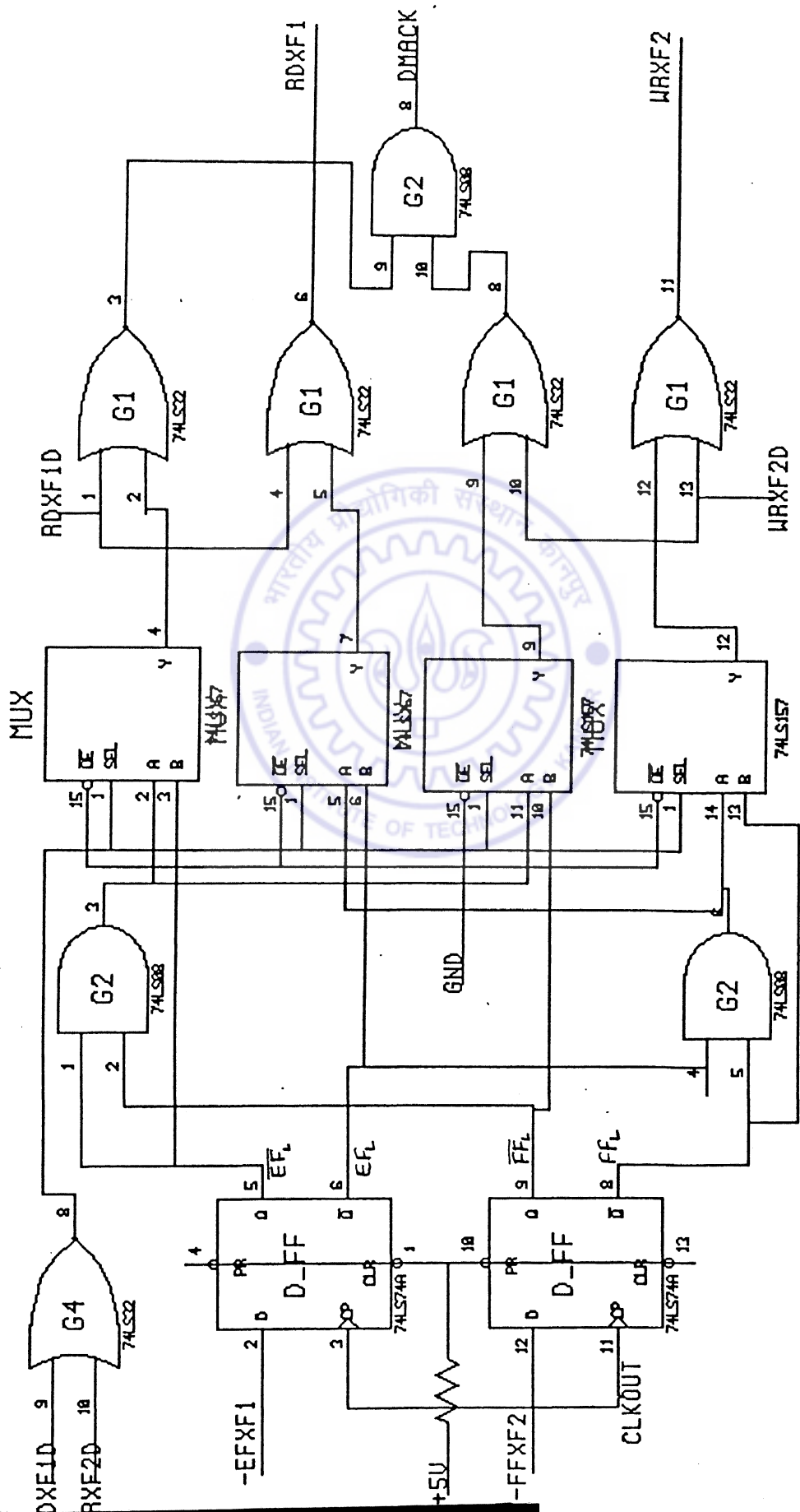


Fig. 5.6 Circuit for X-Channel Flow Control.

fed to XF1 and XF2 (i.e. $\overline{\text{RDXF1}}$ and $\overline{\text{WRXF2}}$ resp.) are the ones that are the modified versions of corresponding signals ($\overline{\text{RDXF1D}}$ and $\overline{\text{WRXF2D}}$ resp.) coming out of EPLDX. Thus, when the $\overline{\text{EFXF1}}$ or $\overline{\text{FFXF2}}$ are high, the reading and writing to these FIFOs go as normal. But whenever any of these flag signals get asserted, this additional circuitry automatically generates appropriately delayed versions of the read/write signals to these FIFOs. The MUXes provided in this circuit takes care of that data transfer in which the data read from XF1 and data write to XF2 takes place in the same instruction cycle. Under such a data transfer condition, neither of the two signals $\overline{\text{EFXF1}}$ or $\overline{\text{FFXF2}}$ should be low for a normal data transfer. Whenever any of these signal is low, the processor should go into the wait state. This is what the set of MUXes is precisely doing by making the assertion of these signals ($\overline{\text{EFXF1}}$ or $\overline{\text{FFXF2}}$) virtually look like as if both of them have been asserted, and so both the read pulse to XF1 as well as the write pulse to XF2 are appropriately and equally delayed.

5.3.2 Y-Channel Flow Control :

For monitoring the empty flag ($\overline{\text{EF}}$) and full flag ($\overline{\text{FF}}$) signals of the FIFOs on the Y-channel, the four interrupts of the processor are utilized. The $\overline{\text{EF}}$ of the forward YFIFO (i.e. YF1) and $\overline{\text{FF}}$ of the reverse YFIFO (i.e. YF1D) on that PE, and $\overline{\text{EF}}$ of the reverse YFIFO (i.e. YF2D) and $\overline{\text{FF}}$ of the forward YFIFO (i.e. YF2) on the right PE are all continuously latched by the trailing edge of the CLKOUT signal. Whenever the last valid read (write) has been carried out on any of these YFIFOs, the $\overline{\text{EF}}$ ($\overline{\text{FF}}$) of that YFIFO

goes low and the same information gets latched by the very next trailing edge of the CLKOUT signal. Now when the next read (write) is tried on that YFIFO, the corresponding interrupt is activated. Once into the interrupt service subroutine, the status of that flag is continuously checked, till it gets de-asserted, after which the aborted instruction is repeated and then the return from that interrupt routine is affected. The circuitry for this part is given in Fig. 5.8 and the corresponding timing diagram is depicted in Fig. 5.9. It is to be noted that ADSP 2100 branches to the interrupt service routine after an overhead of two instructions, i.e. not only the instruction during which the interrupt came is executed, the subsequent instruction also gets executed before it branches to interrupt service subroutine. So each read/write instruction from/to YFIFO should be followed either by a NOP instruction in the main program, or alternatively in the interrupt service subroutine, the aborted read/write instruction together with the instruction that follows it should be repeated before the control returns back from the subroutine.

5.4 Processing Element Card Assembly :

The schematic diagram of the PE card is given in Appendix C. The whole circuitry is assembled on a 15" X 9" double-layered PCB, which plugs into the backplane PCB through two 96-pin Euroconnectors. The total chip count is 50. All 2K memories are Motorola MCM2018a (2K X 8) chips, and all 8K memories are Motorola MCM6264 (8K X 8) chips. The FIFOs are implemented with Inmos MK45H01 (1K X 9 bits). The physical placement of the chips on the

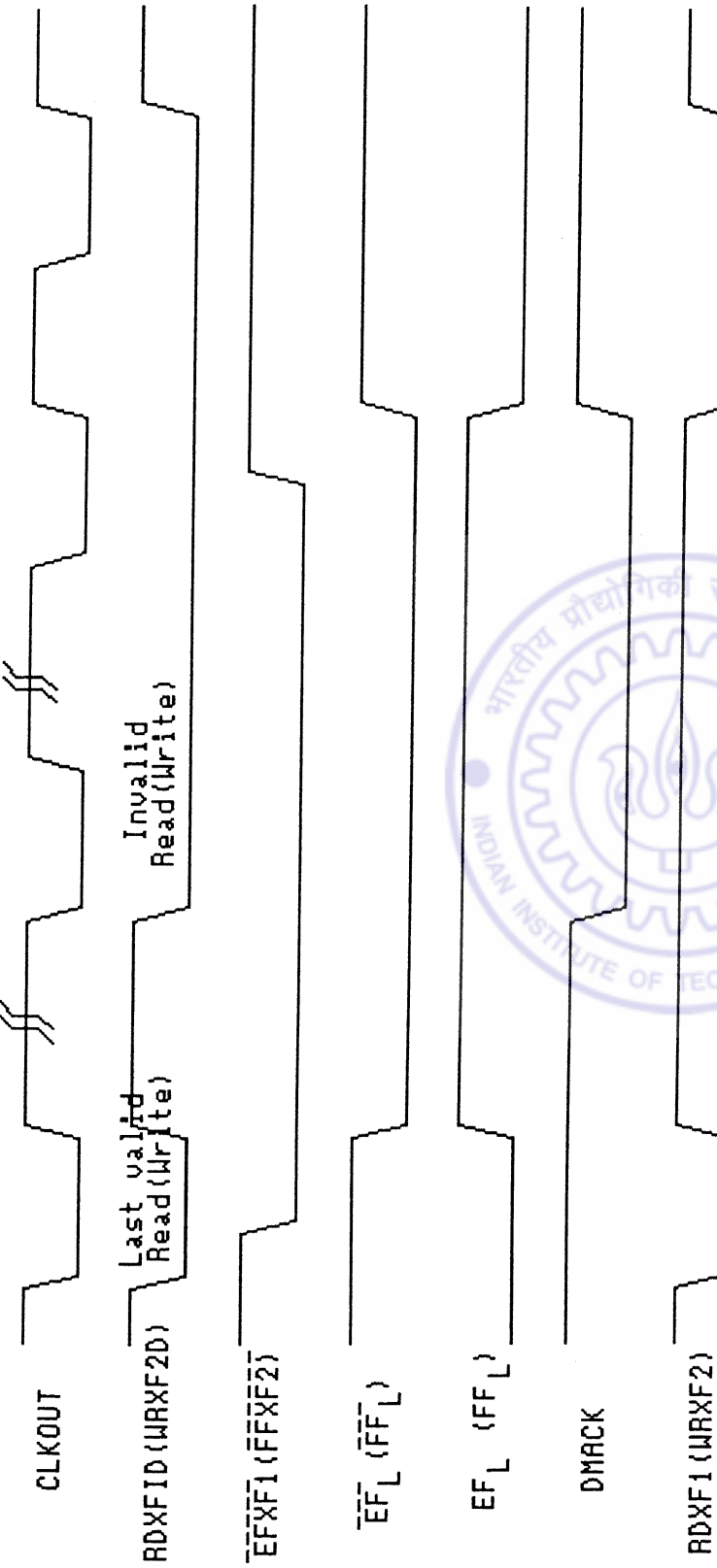


FIG 5.7 TIMING DIAGRAM FOR X-CHANNEL FLOW CONTROL.

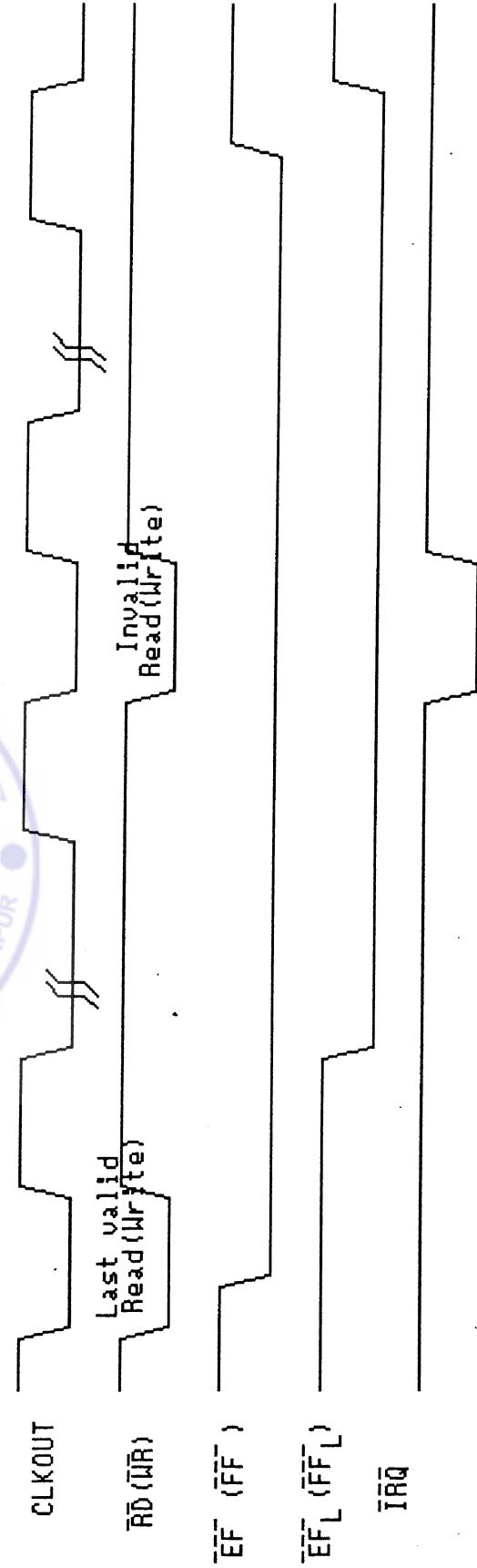


FIG 5.9 TIMING DIAGRAM FOR Y-CHANNEL FLOW CONTROL .

PCB is shown in Appendix D. The pin assignment for the Euroconnectors is given below :

Euroconnector 1 :

Pin 1	: XD ₁₄
Pin 2	: XD ₁₅
Pin 3 to Pin 32	: Not connected
Pin 33	: $\overline{\text{MEMR}}$
Pin 34	: $\overline{\text{MEMW}}$
Pin 35	: SBHE
Pin 36	: $\overline{\text{RESET}}$
Pin 37	: $\overline{\text{BR}}$
Pin 38 to Pin 40	: PESL, PESM, PESU.
Pin 41	: $\overline{\text{BC}}$
Pin 42	: $\overline{\text{HALT}}$
Pin 43	: MS
Pin 44	: MASTERCLK
Pin 45	: AEN
Pin 46 and Pin 47	: Not connected
Pin 48	: Vcc (+5V)
Pin 49	: Ground (GND)
Pin 50	: Not connected
Pin 51 to Pin 64	: XD ₀ to XD ₁₃
Pin 65 to Pin 80	: SD ₀ to SD ₁₅
Pin 81 to Pin 96	: SA ₀ to SA ₁₅

Euroconnector 2 :

Pin 1	: WRXF1
Pin 2	: $\overline{\text{FFXF1}}$

Pin 3 to Pin 32 : Not connected
 Pin 33 : $\overline{\text{BGHI}}$
 Pin 34 : HTRAPI
 Pin 35 : $\overline{\text{MEMCS16I}}$
 Pin 36 : WRYF2
 Pin 37 : RDYF2D
 Pin 38 : $\overline{\text{FFYF2}}$
 Pin 39 : $\overline{\text{EFYF2D}}$
 Pin 40 : WRXF2
 Pin 41 : $\overline{\text{FFXF2}}$
 Pin 42 to Pin 57 : YD₀ to YD₁₅
 Pin 58 : $\overline{\text{BGHO}}$
 Pin 59 : HTRAPO
 Pin 60 : $\overline{\text{MEMCS16O}}$
 Pin 61 : WRYF1
 Pin 62 : RDYF1D
 Pin 63 : $\overline{\text{FFYF1}}$
 Pin 64 : $\overline{\text{EFYF1D}}$
 Pin 65 to 80 : DMD₀ to DMD₁₅
 Pin 81 to 96 : PMD₈ to PMD₂₃

CHAPTER 6

DISCUSSIONS AND CURRENT STATUS OF THE WORK

As discussed in the preceding chapters, the design of our circuit has been accomplished upto the PCB level. The PCBs for the host interface, the processing element and the backplane have been manufactured. We populated the interface card and then plugged it into an I/O slot of the host. On switching the PC on, it did not boot. By analyzing, we found out that it were the data buffers which were creating the problem. These permanently enabled buffers were being controlled for their direction by the $\overline{\text{MEMR}}$ signal of the host. So, whenever the host was into any memory read cycle whatsoever, the floating outputs of these data buffers created the bus contention problem on the PC-data bus. This problem had to be circumvented by qualifying the control of these buffers by a master chip select ($\overline{\text{MCS}}$) for the processor array.

But before that, we disabled the data buffers permanently and then tested the I/O operation of our interfacing card. It was found that the control register was being written and the status register being read by the host. The control register outputs were then fed back to the status register and a series of data-writes into the control register and the subsequent data-reads from the status register were successfully performed. Moreover, the

data-write signal to the other I/O port (IO1)-the XFIFO of the first PE-was also being generated. This was tested by programming the host to issue (in an infinite loop) the write signals to the port IO1 and the same was viewed on the CRO. The same exercise was repeated for a TTL load (~2K) as the I/O port and the voltage levels were found to be satisfactory. But for the slight ringing, the waveform of the data write signal to the port IO1 was found to be satisfactory. In this way, it may be concluded that the host interfacing was successfully performing the I/O operation.

Now for solving the data-buffer problem, some additional circuitry had to be mounted on the interface card. For this, the interface card was extended and two 4-bit comparators (74LS85), an inverter (74LS04) and a DIP switch were included in the interface. Their function was to decode the SA₁₆ to SA₁₉ address lines of the host. The AEN signal of the host was also taken care of here itself. The schematic for this circuitry is given in Fig.6.1. The active-high output of the comparator (COMP2) is inverted to get the Master Chip Select (\overline{MCS}). Now this \overline{MCS} signal is used to enable the data buffers on the interface card. Moreover, it is this \overline{MCS} signal which is broadcasted instead of AEN from the host to all the PEs for the generation of chip selects for their memories. Then using the Debug command in DOS, we found that the memory slot A0000H to A3FFFH is unused in our host. So we tried to map our processor array into this 16 KB slot of host memory. We tried the loading of

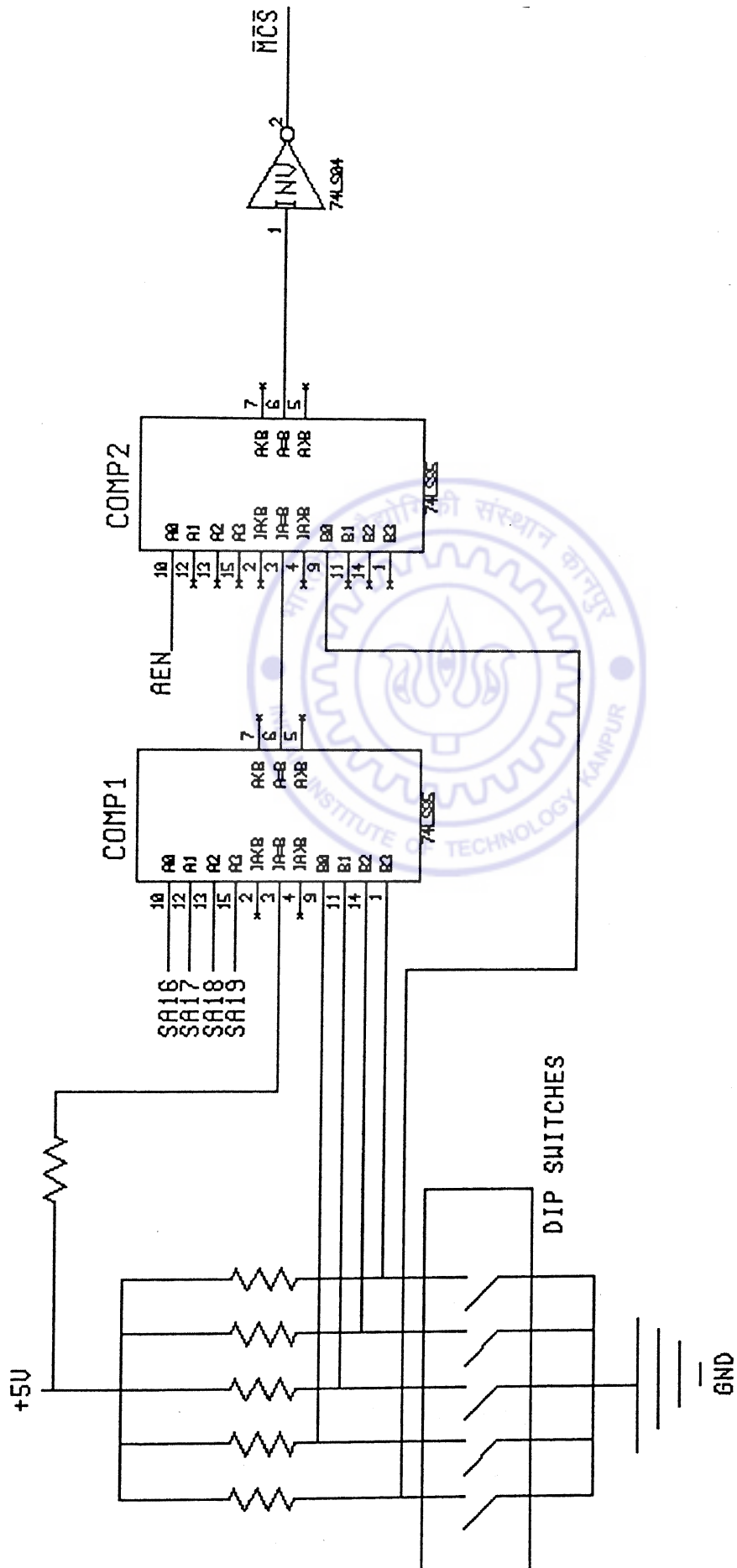


FIG 6.1 MASTER CHIP SELECT GENERATION.

the PE memories from the host. We faced two problems. It was observed that whenever the word-wise loading is being tried to either PMM and PMU, or DML and DMU, or PMDML and PMDMU, the data line SD₁₀ seemed to be tied to logic 1. Also for the byte-wise loading to PML, the loading is satisfactory when the address line SA₁₁ is 1, but as soon as it becomes zero, two consecutive locations get loaded by the same data. We tried to further analyze these problems by putting alternately zeros and ones on the address lines SA₀ and SA₁₁ in an infinite loop, and viewing the waveforms of the two address lines on the CRO. We observed that when any of the two address lines SA₀ or SA₁₁ goes to zero, it pulls the other line also to zero along with it. This clearly showed that these two address lines are shorted somewhere. The short was searched, removed and the up- and down-loading of all the memories on the PE card was satisfactorily accomplished. Due to lack of time, the testing process could not be pursued further. For complete performance evaluation of the system, following tests may be carried out :

- * accessing of these memories from the processor ADSP 2100
- * extending the above tests to more than one PE.
- * checking for the various interprocessor communication options.
- * run-time flow control testing on both X-channel and Y-channel.
- * performance evaluation of the array by mapping a few signal and image processing algorithms onto it.

REFERENCES

- [1] S. Y. Kung, "VLSI Array Processors", Prentice Hall, 1988.
- [2] Ioannis Pitas, "Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks", John Wiley, 1993.
- [3] H. T. Kung, "Why Systolic Architectures?", IEEE Computer, vol.15, no.1, Jan. 1982.
- [4] M. Annaratone et al, "The Warp Computer Architecture Implementation and Performance", IEEE Trans. Computers, vol.36, no.12, pp.1523-1538, Dec. 1987.
- [5] R. Cypher, J. L. C. Sanz, "SIMD Architectures and Algorithms for Image Processing and Computer Vision", IEEE Trans. Acoustics, Speech and Signal Processing, ASSP, vol.37, no.12, pp.2158-2174, Dec. 1989.
- [6] R. K. Sharma, "A Class of Linear Systolic array Using ADSP 2100", M.Tech. Thesis, I.I.T. Kanpur, March 1993.
- [7] S. Bhaishya, "Performace Evaluation of a Linear Array in Low Level Image and Signal Processing Applications", M.Tech. Thesis, I.I.T. Kanpur, Feb.1994.
- [8] J. J. Navarro et al, "Partitioning: An Essential Step in Mapping Algorithms into Systolic Array Processors", Computer Mag, pp.77-89, July 1987.
- [9] H. T. Kung, "Computational Models for Parallel Computers", CMU-CS-88-164, Aug. 28, 1987.
- [10] S. Y. Kung et al, " Wavefront Array Processor - Concept to Implementation", IEEE Computer, pp. 18-33, July 1987.

- [11] H. Burkhardt, B. Lang and M. Noelle, "Aspects of Parallel Image Processing Algorithms and Architectures", In H. Burkhardt, Y. Nevo and J. C. Simon, editors, *From Pixels to Features II*, pp. 65-84, Elsevier, 1992.
- [12] C. Paterson et al, "iWarp: A 100-MOPs LIW Microprocessor for Multicomputers", *IEEE Micro*, p.26, June 1991.
- [13] A. Mahanta, R. K. Sharma, "A Linear Systolic Array Using DSP Microprocessors", *Int. Conf. on SPAT*, 1993.



APPENDIX A

SYSTOLIC ARRAY I

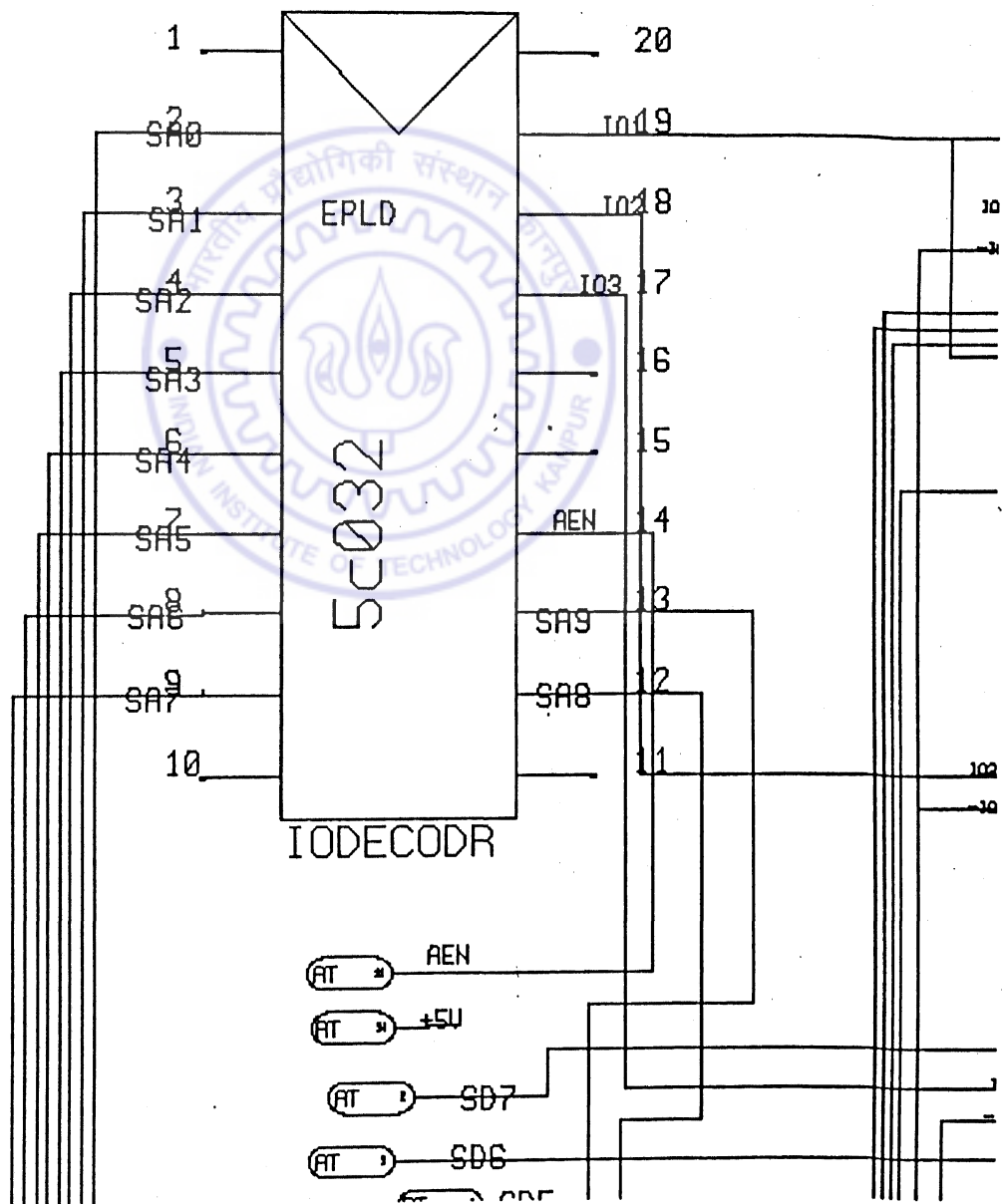
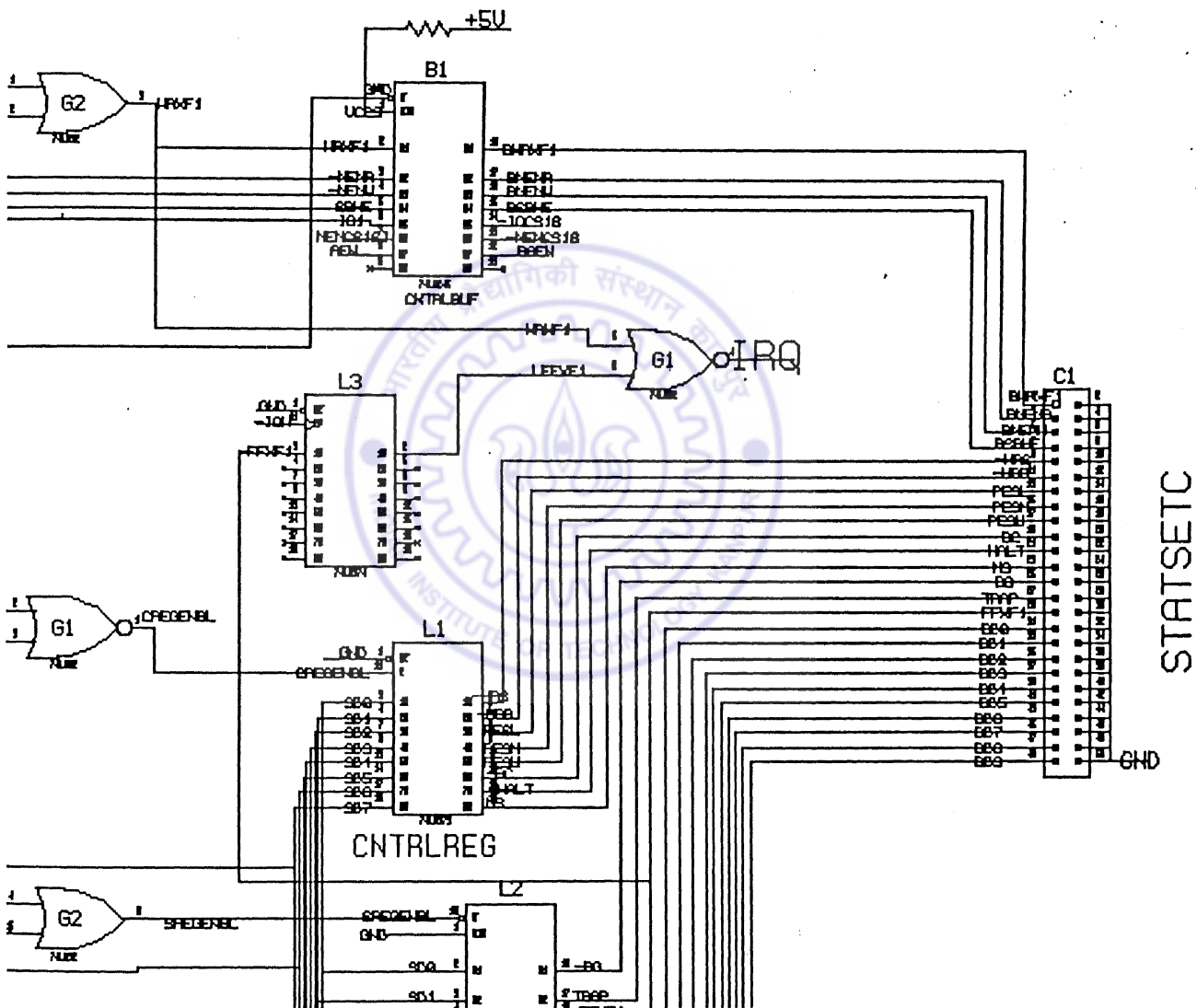


Fig. A.1(a)

INTERFACED TO PC-AT

-M. M. SUFYAN BEG



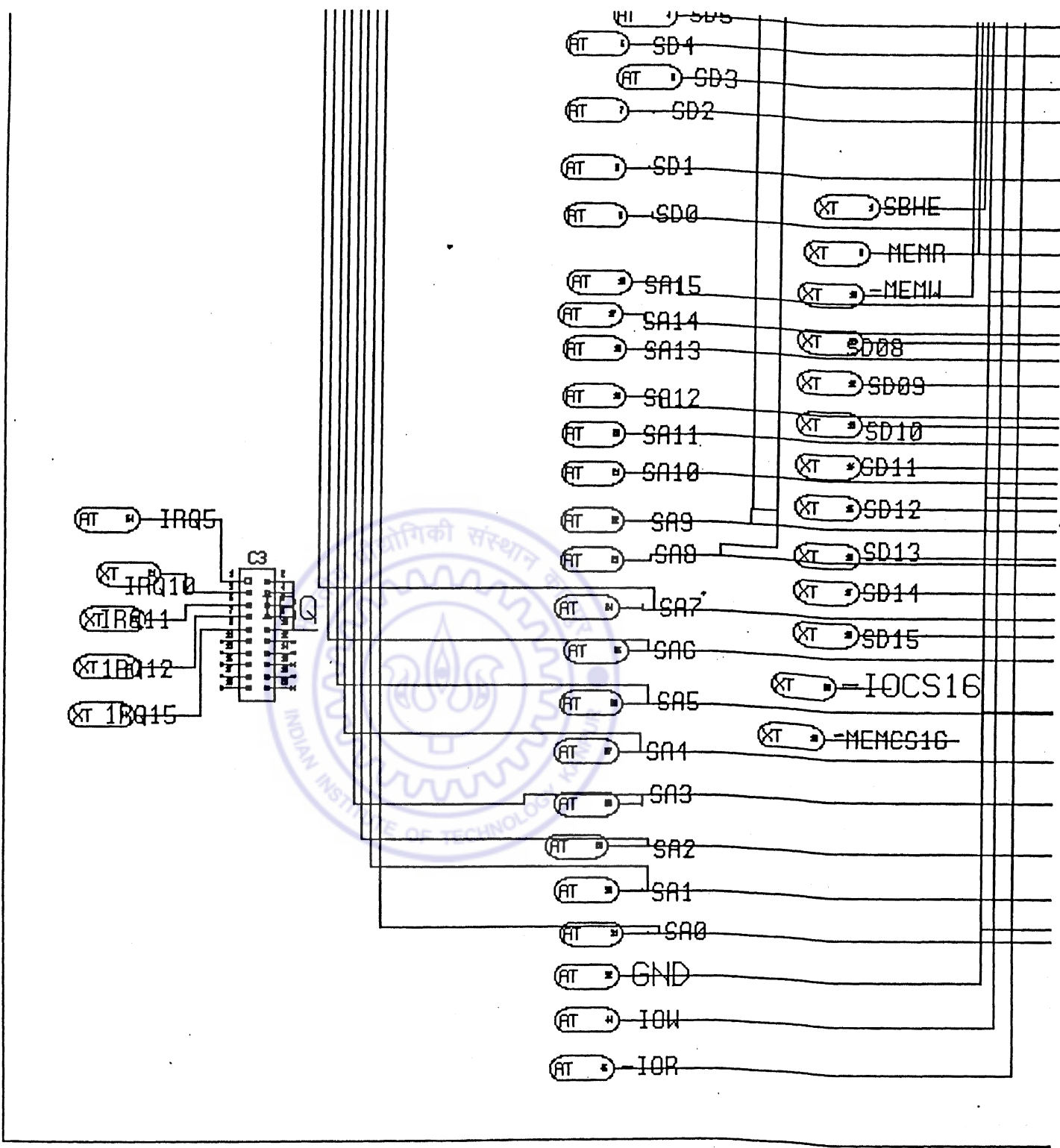


Fig. A.1(c)

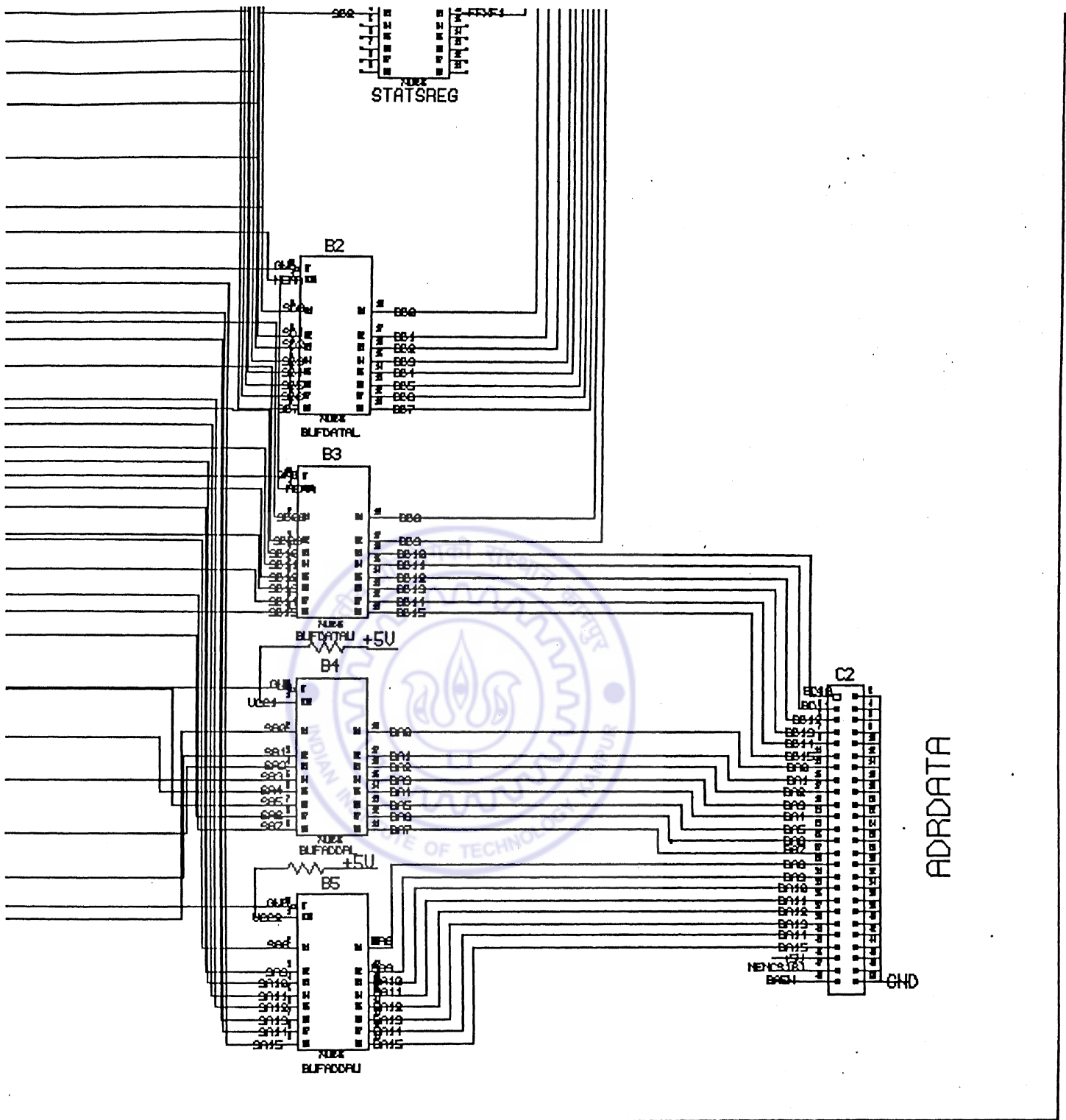


Fig. A.1(d)

APPENDIX B

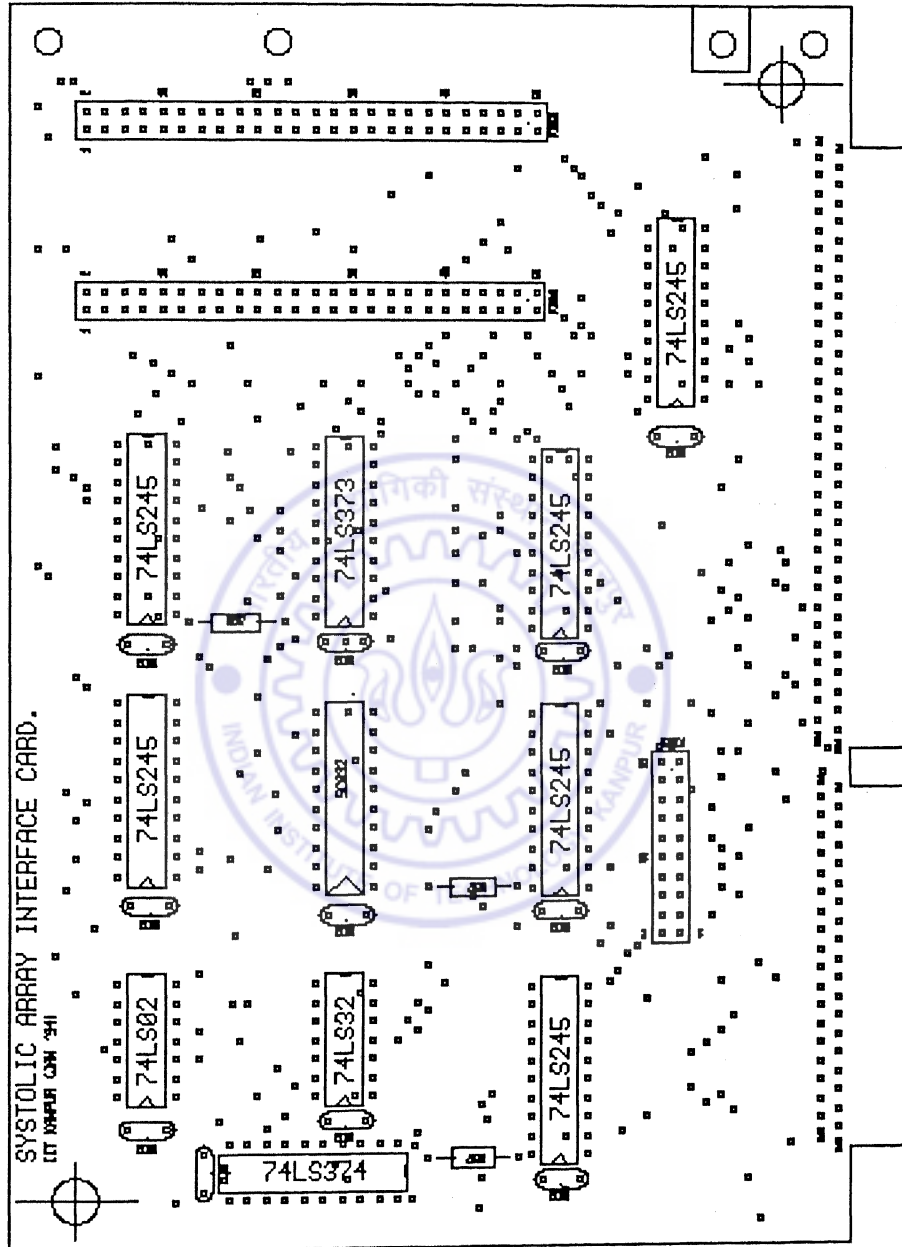


Fig. B.1

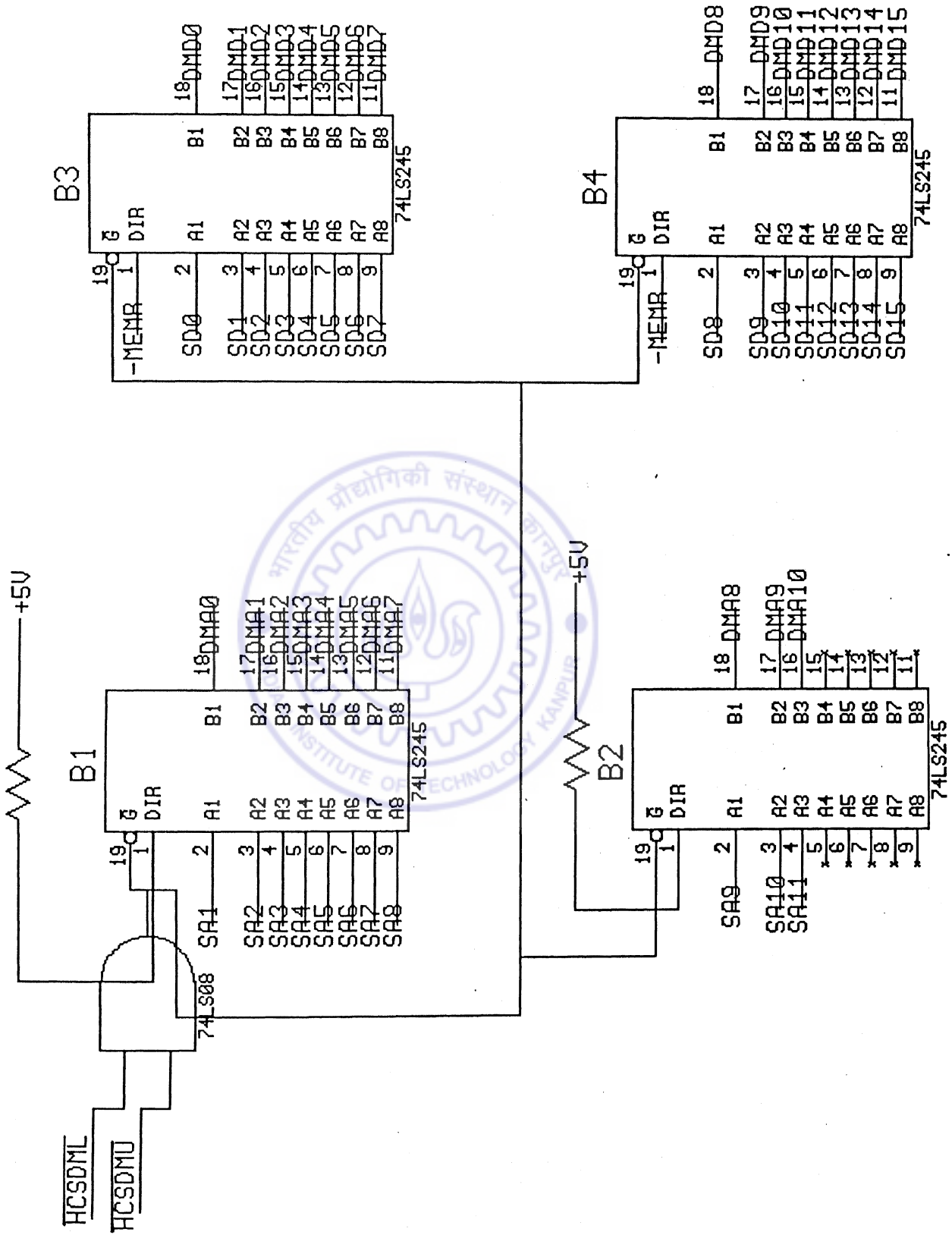


Fig. C.1(a)

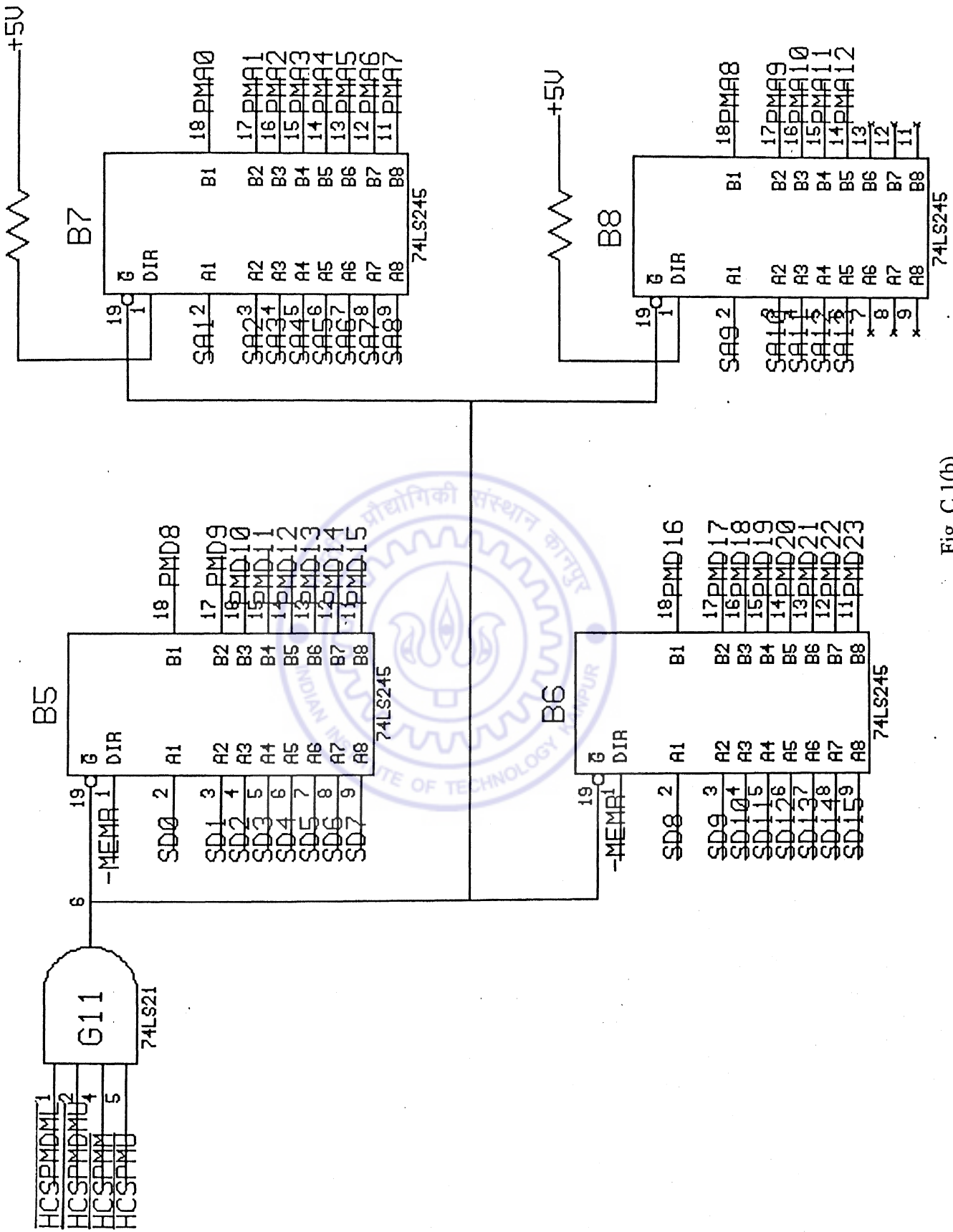


Fig. C.1(b)

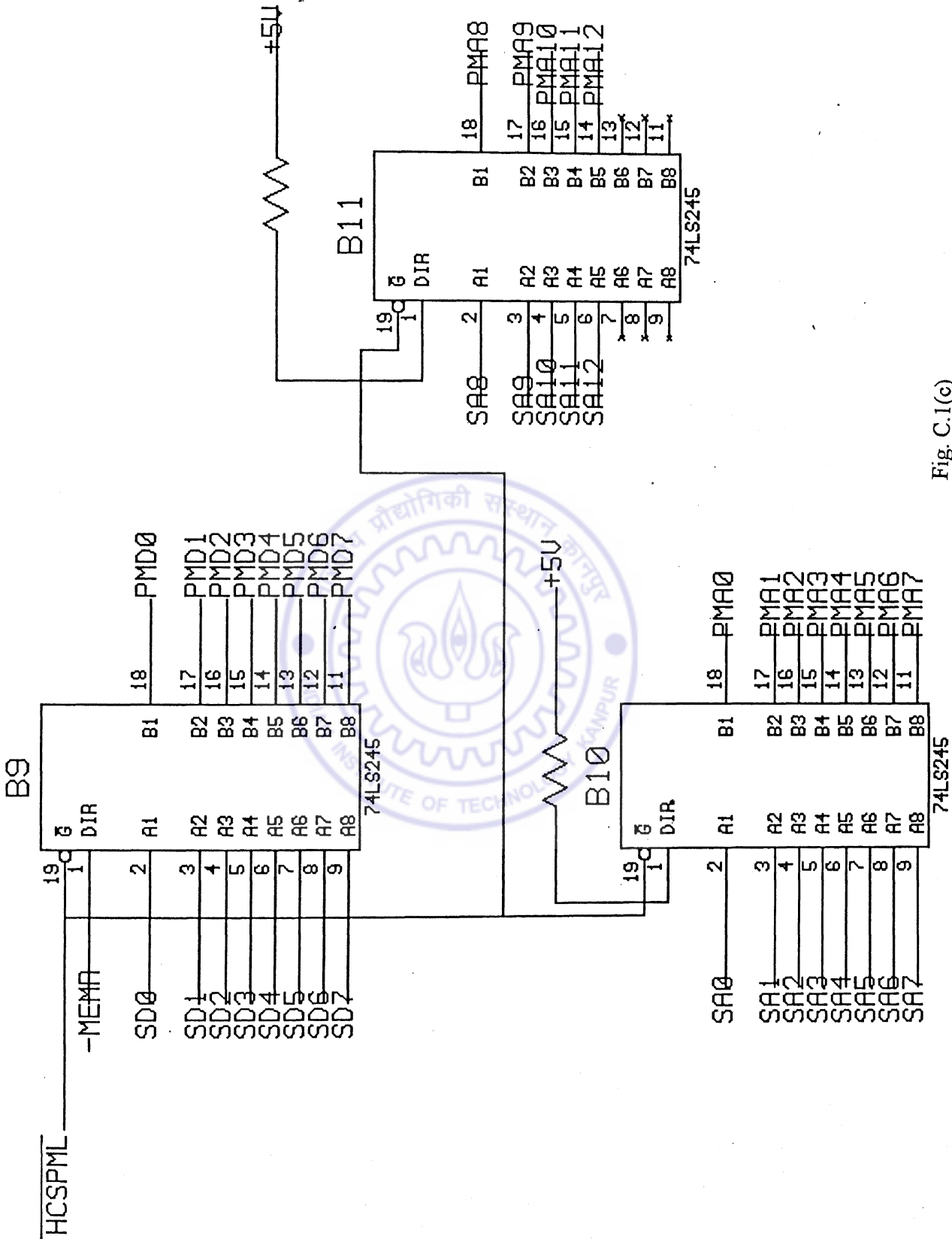


Fig. C.1(c)

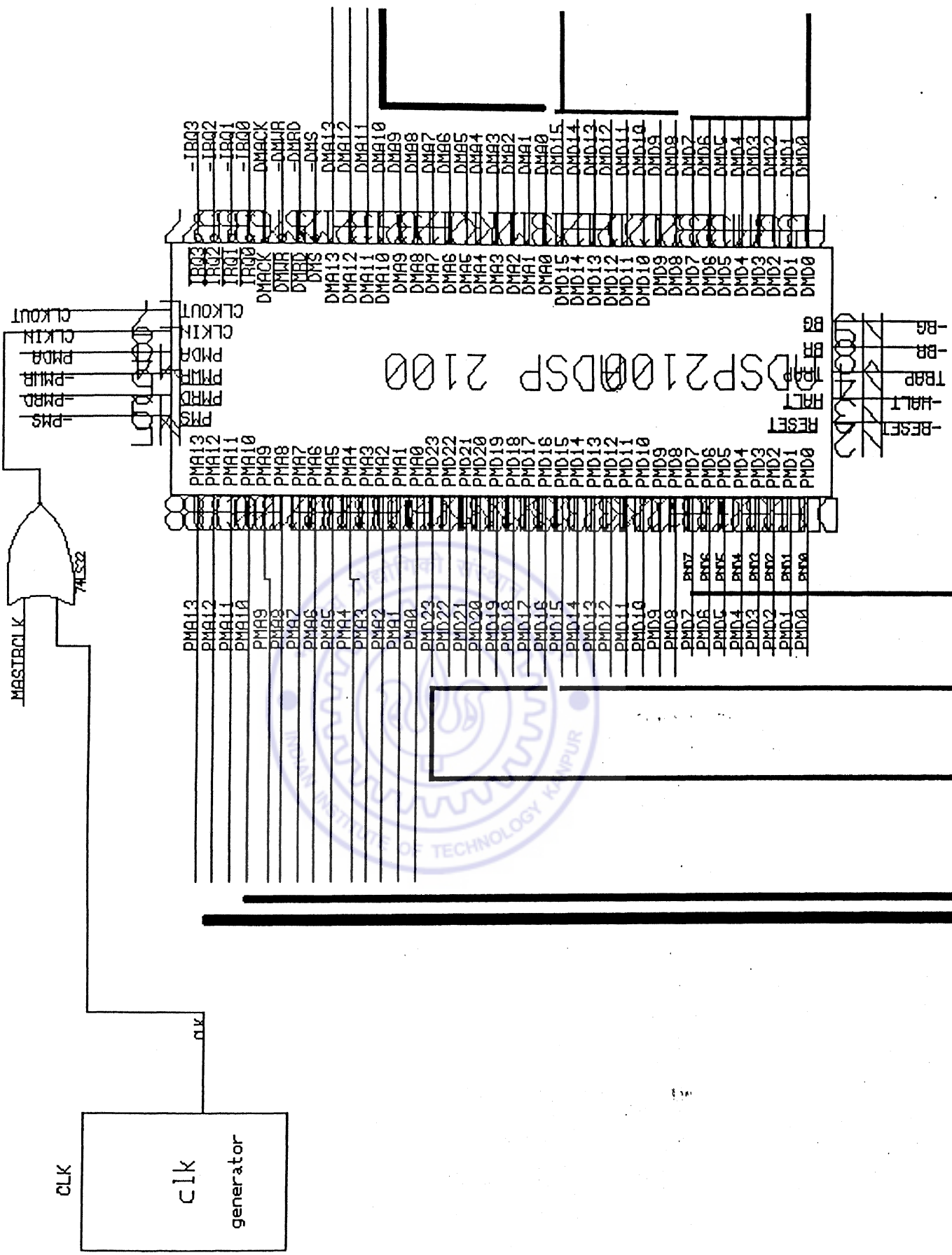


Fig. C.1(d)

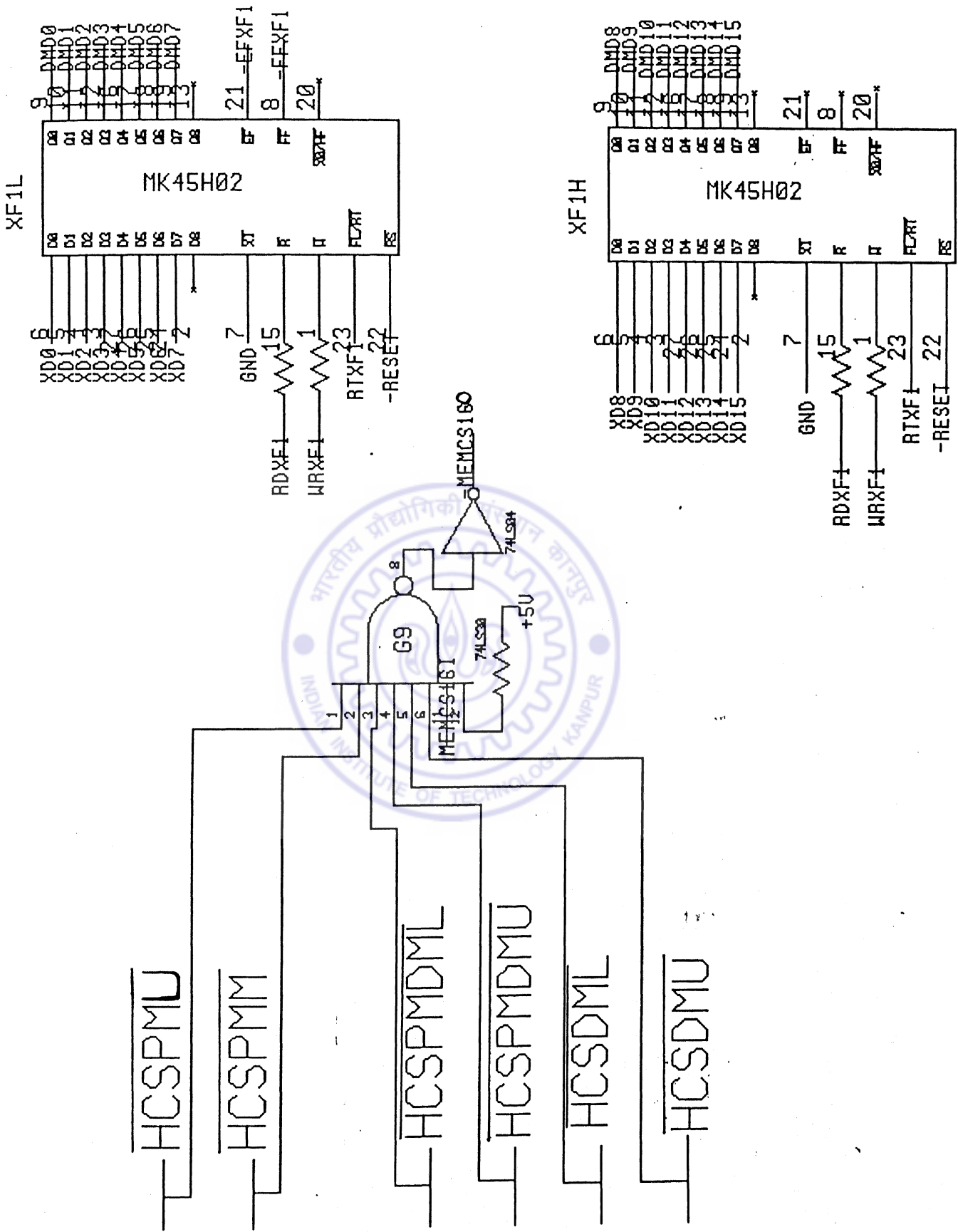


Fig. C.1(e)

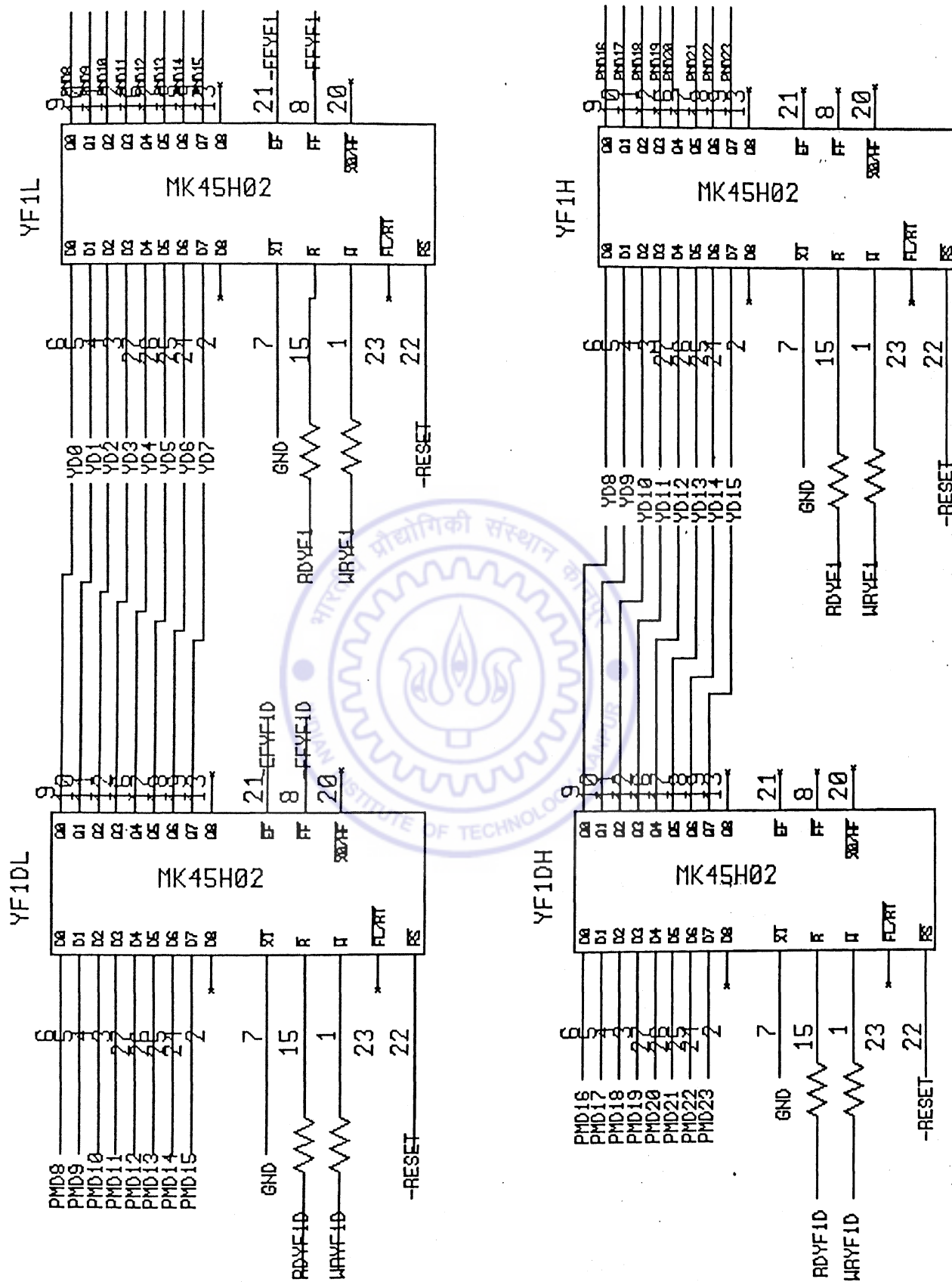


Fig. C.1(f)

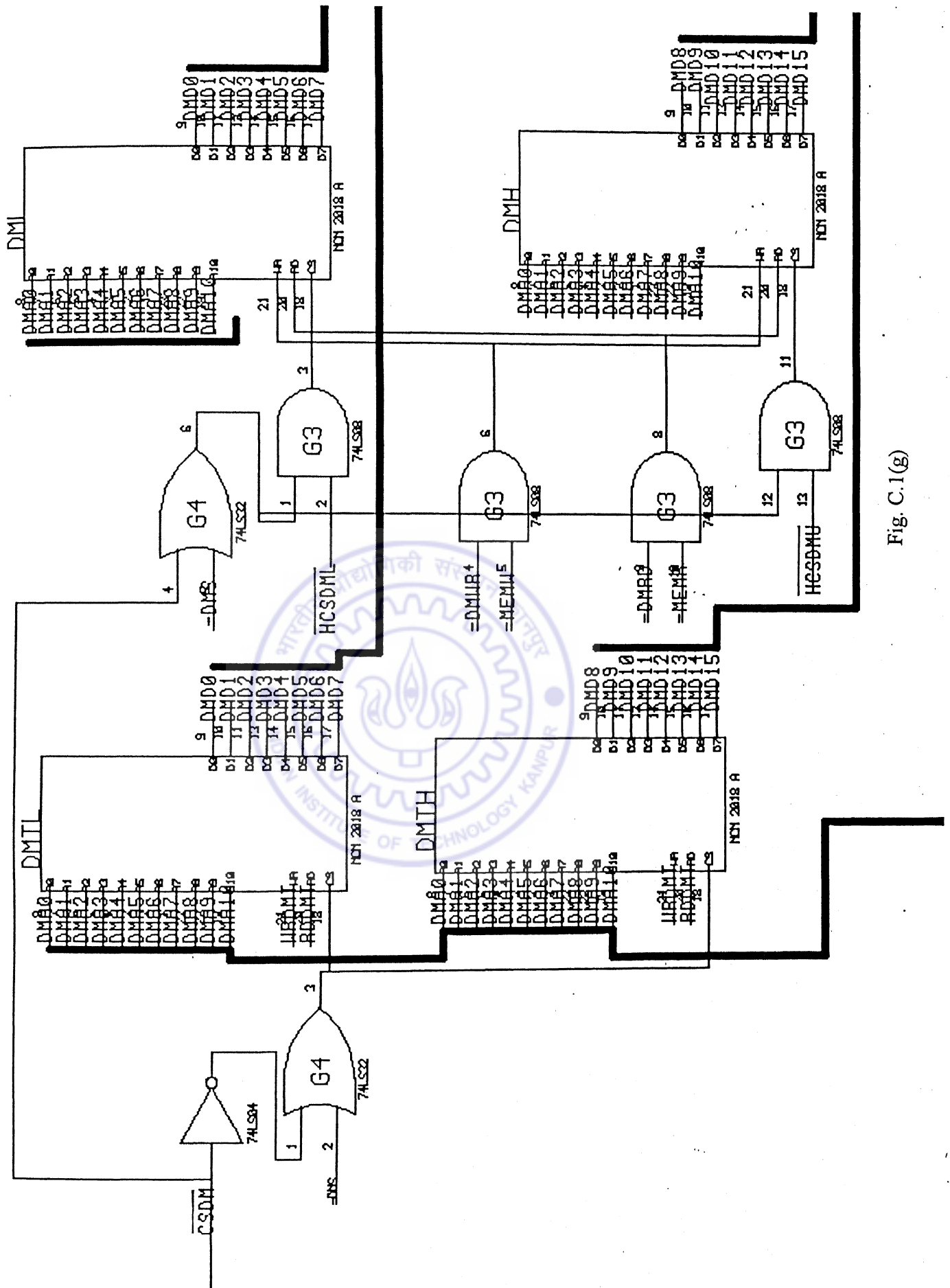


Fig. C.1(g)

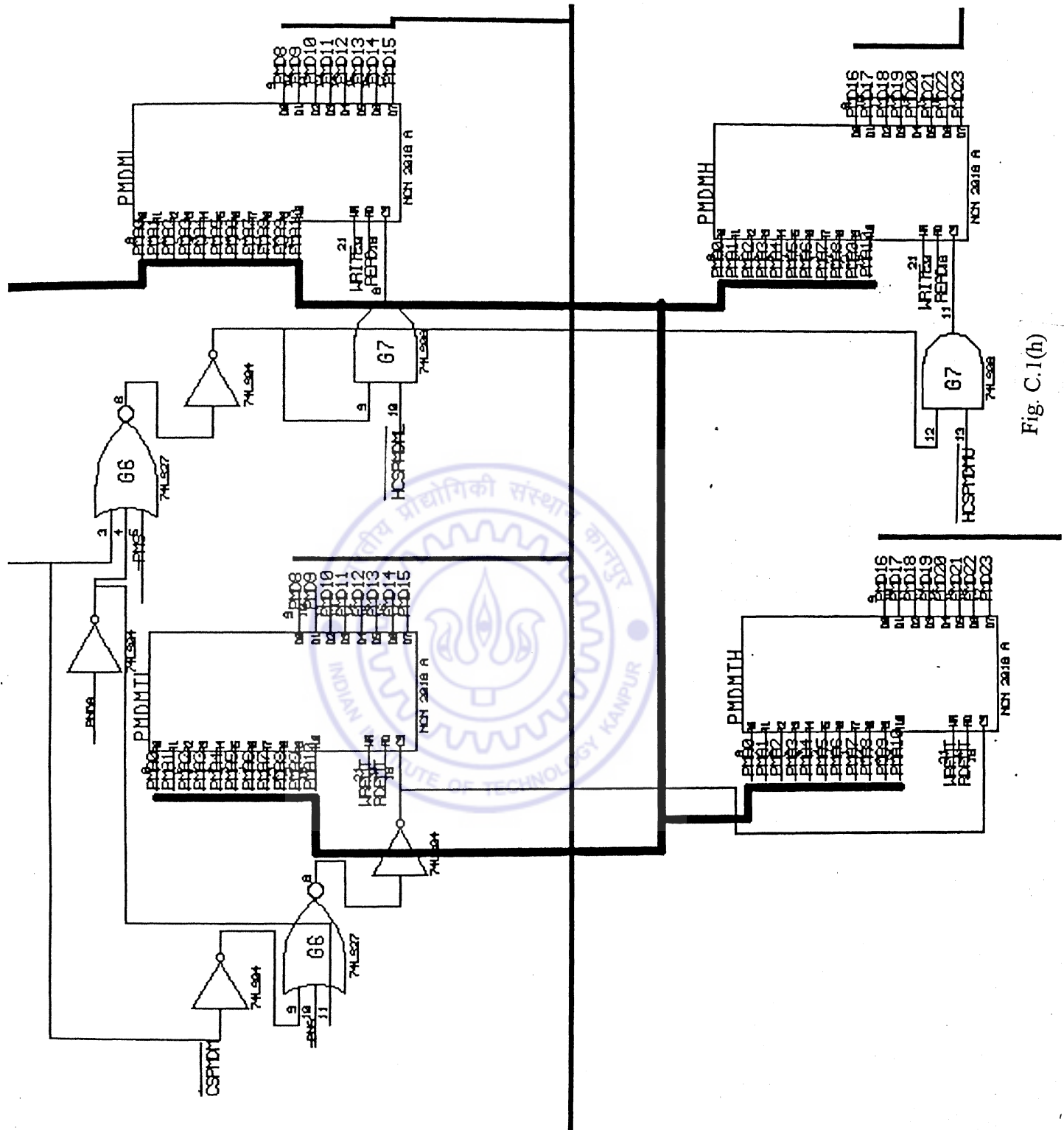


Fig. C.1(h)

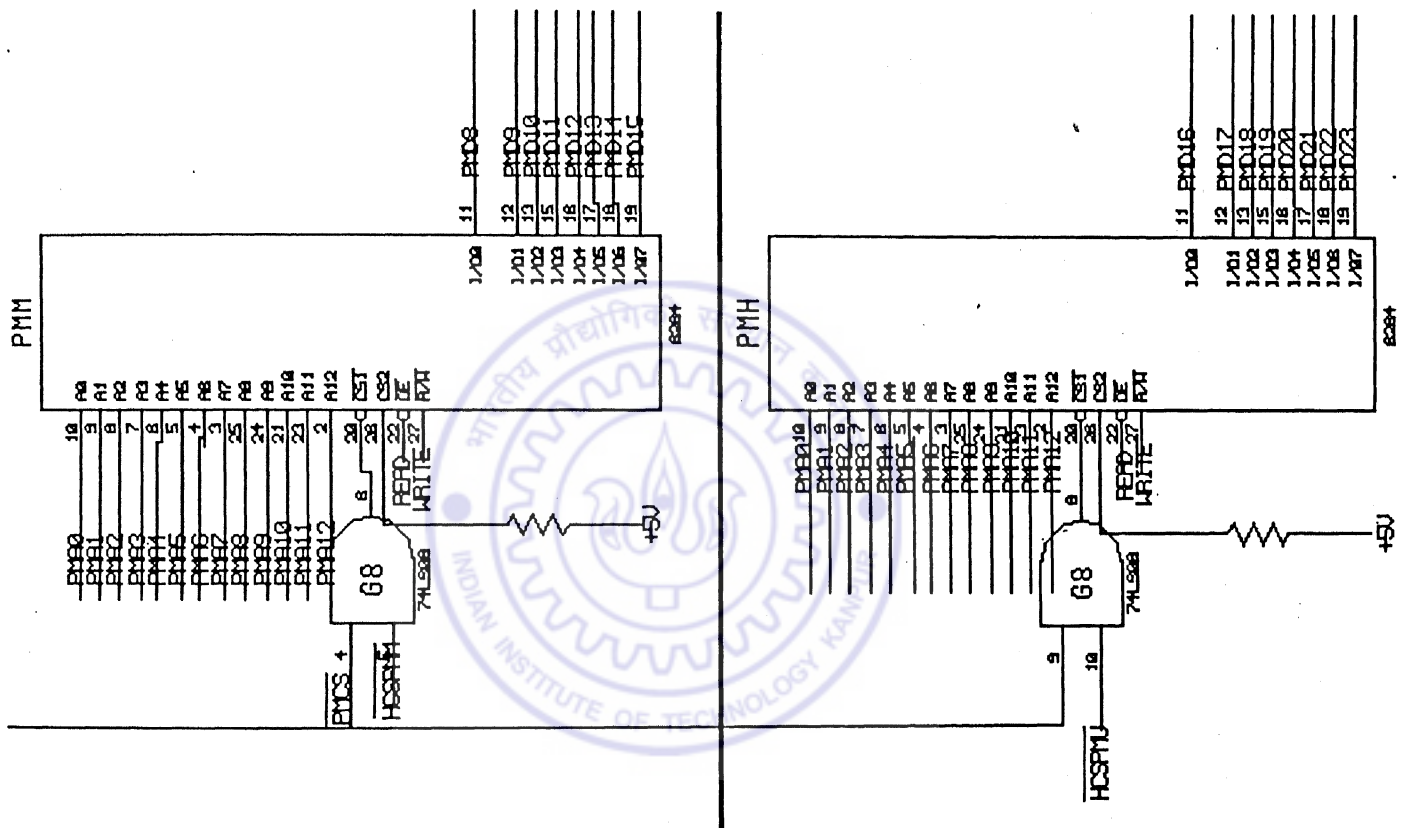


Fig. C.1(i)

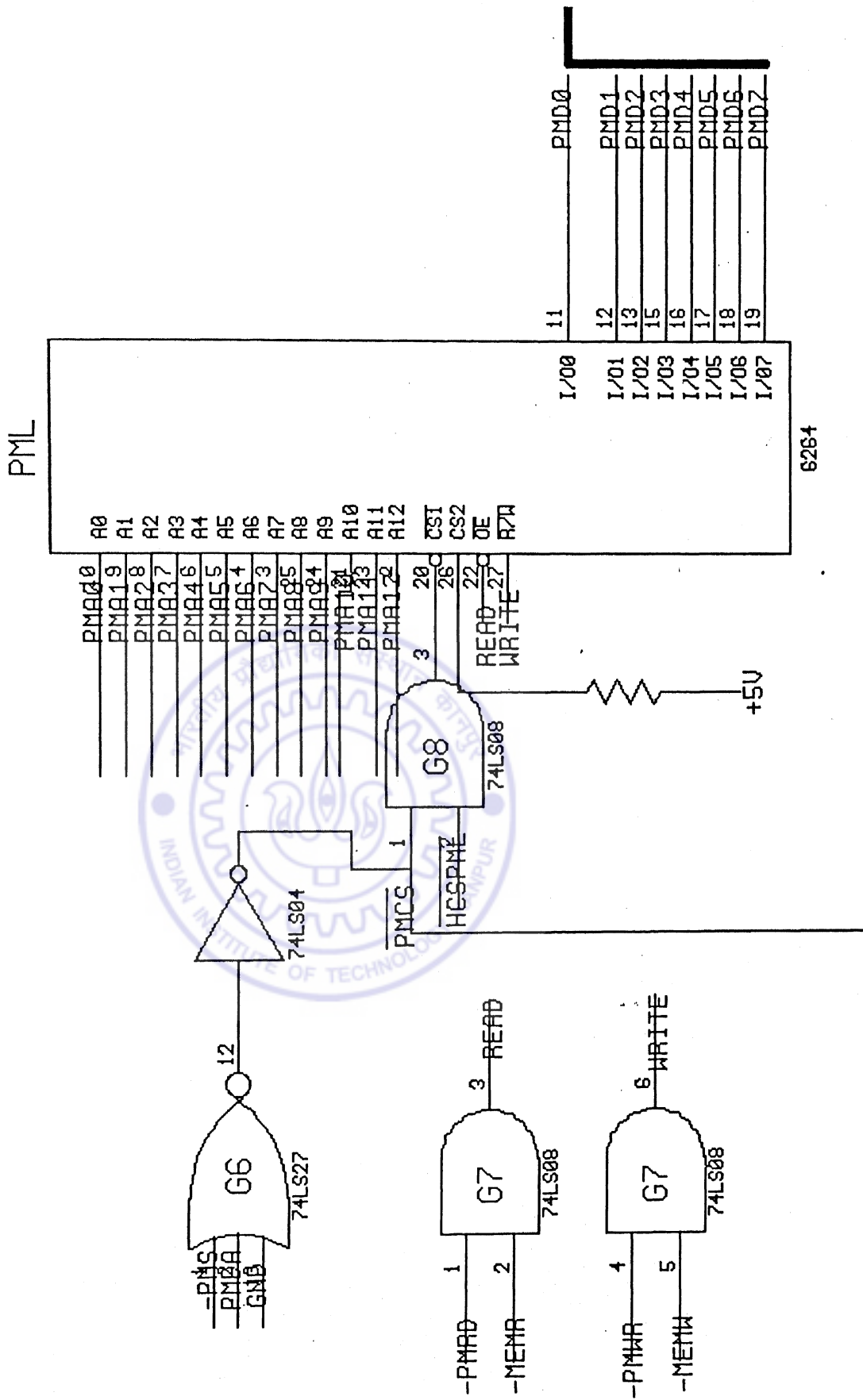


Fig. C.1(i)

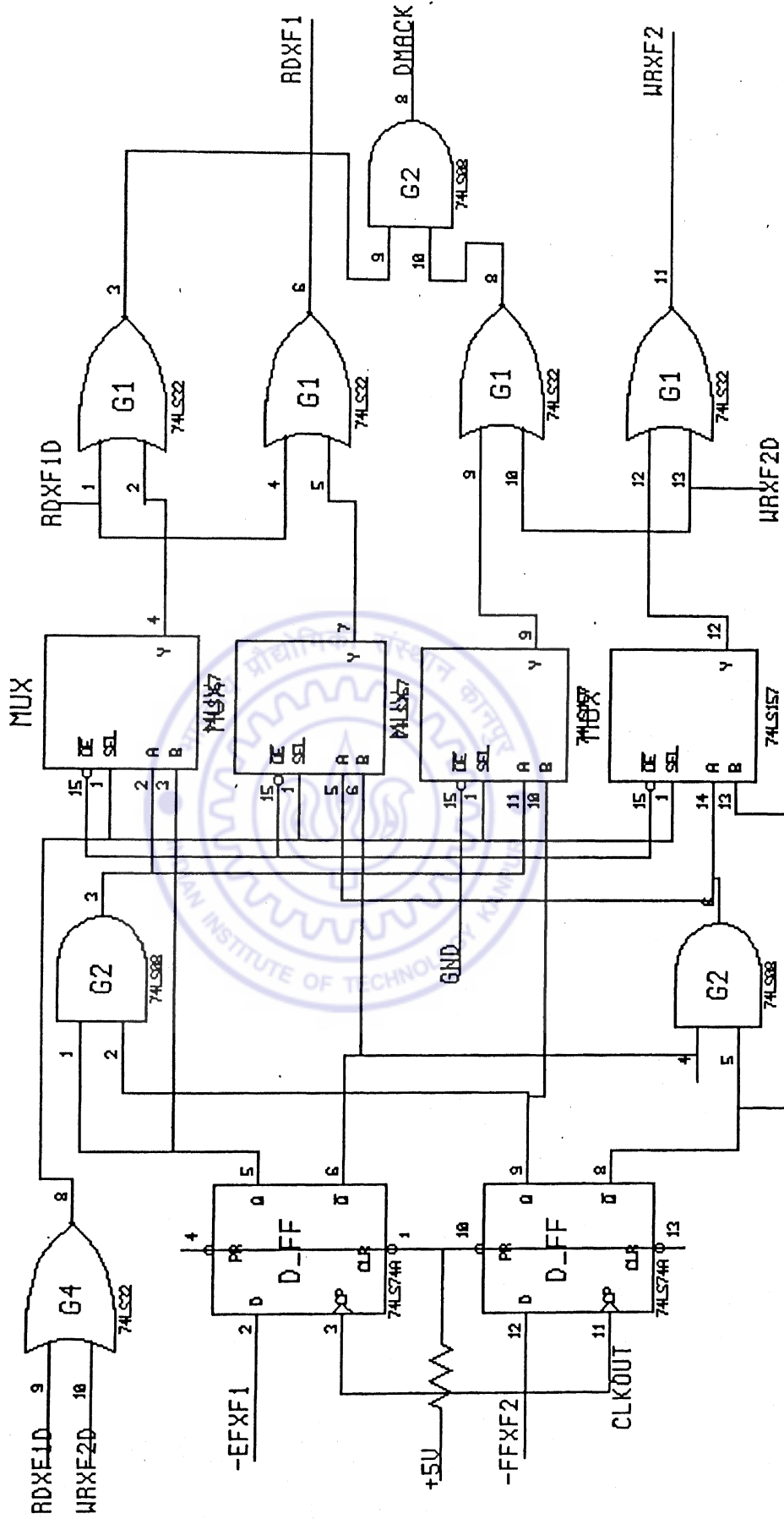


Fig. C.1(k)

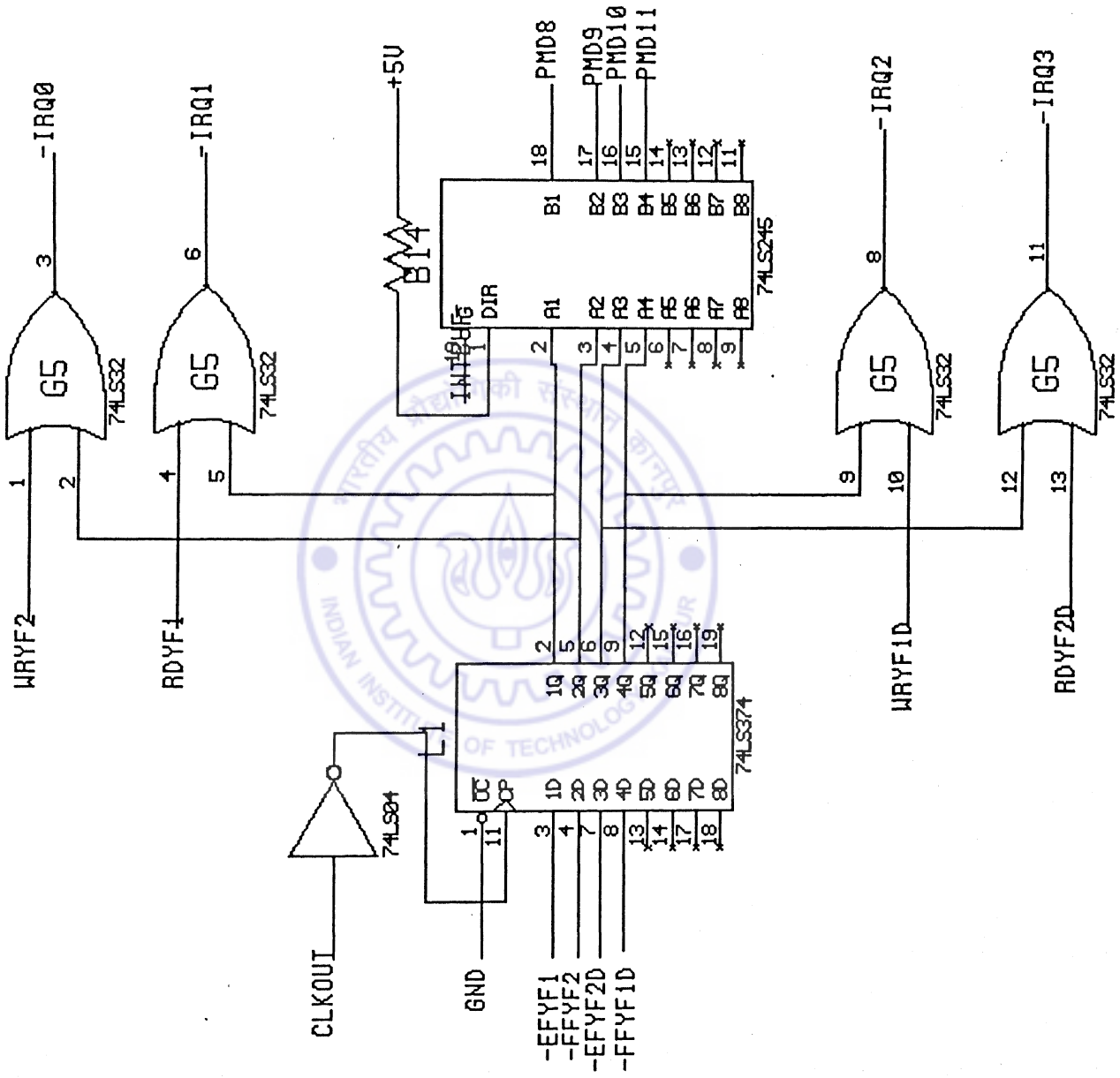


Fig. C.1(i)

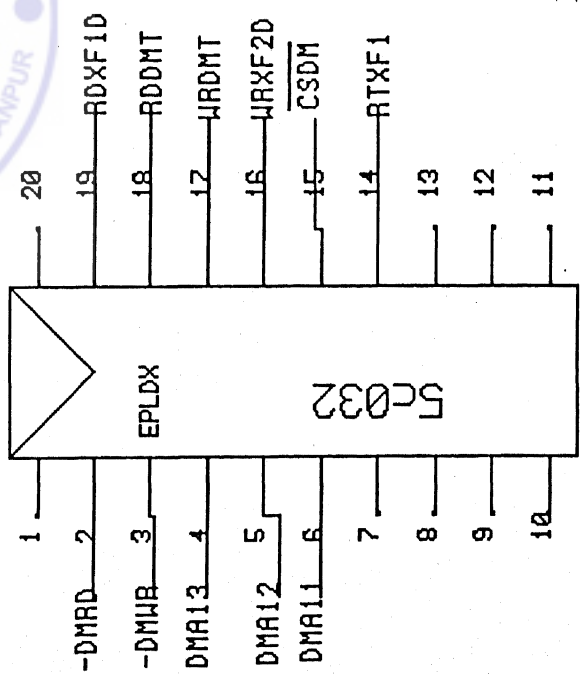
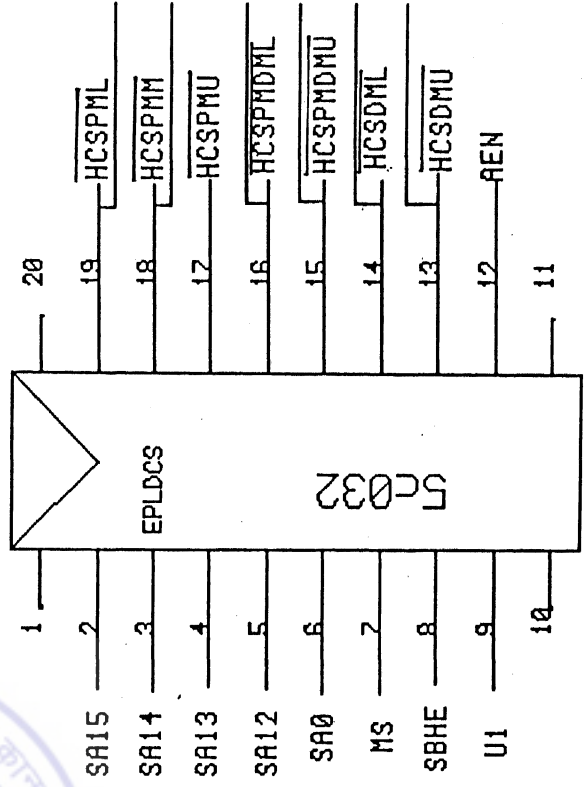
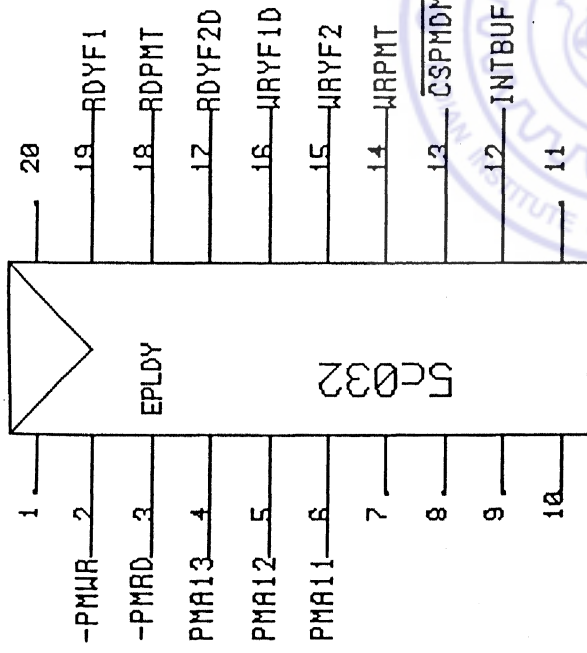
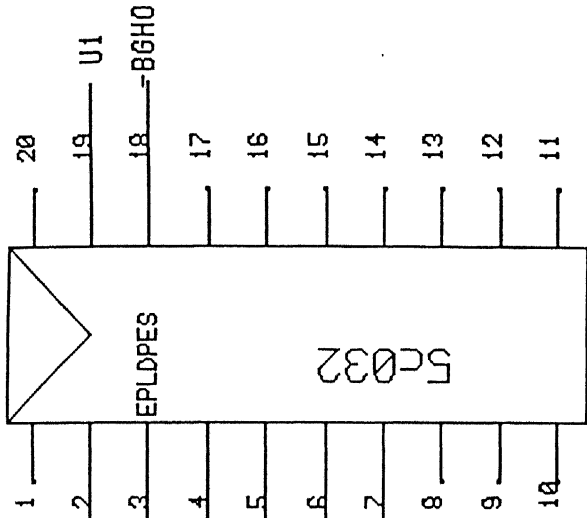


Fig. C.1(m)

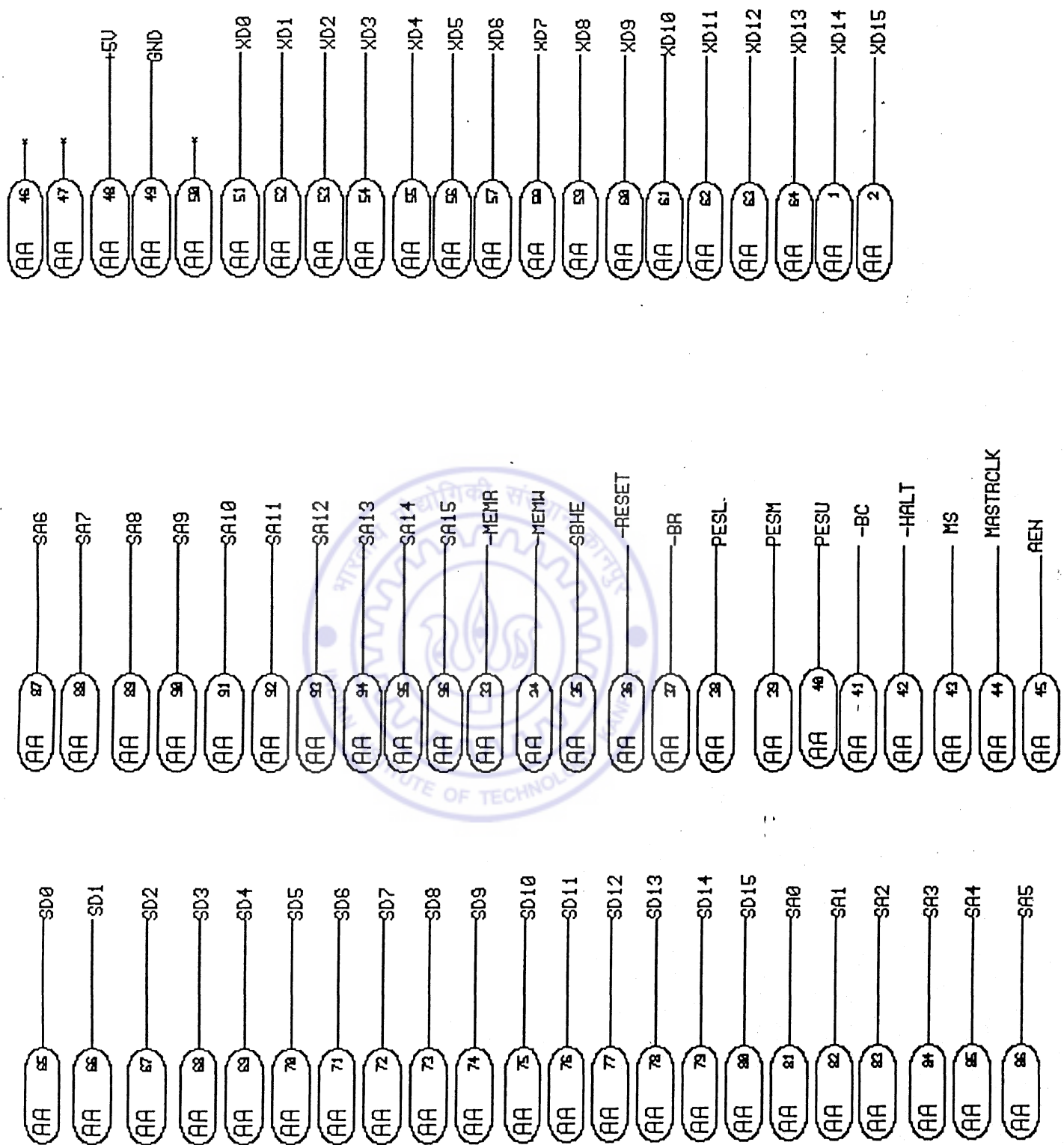


Fig. C.1(n)

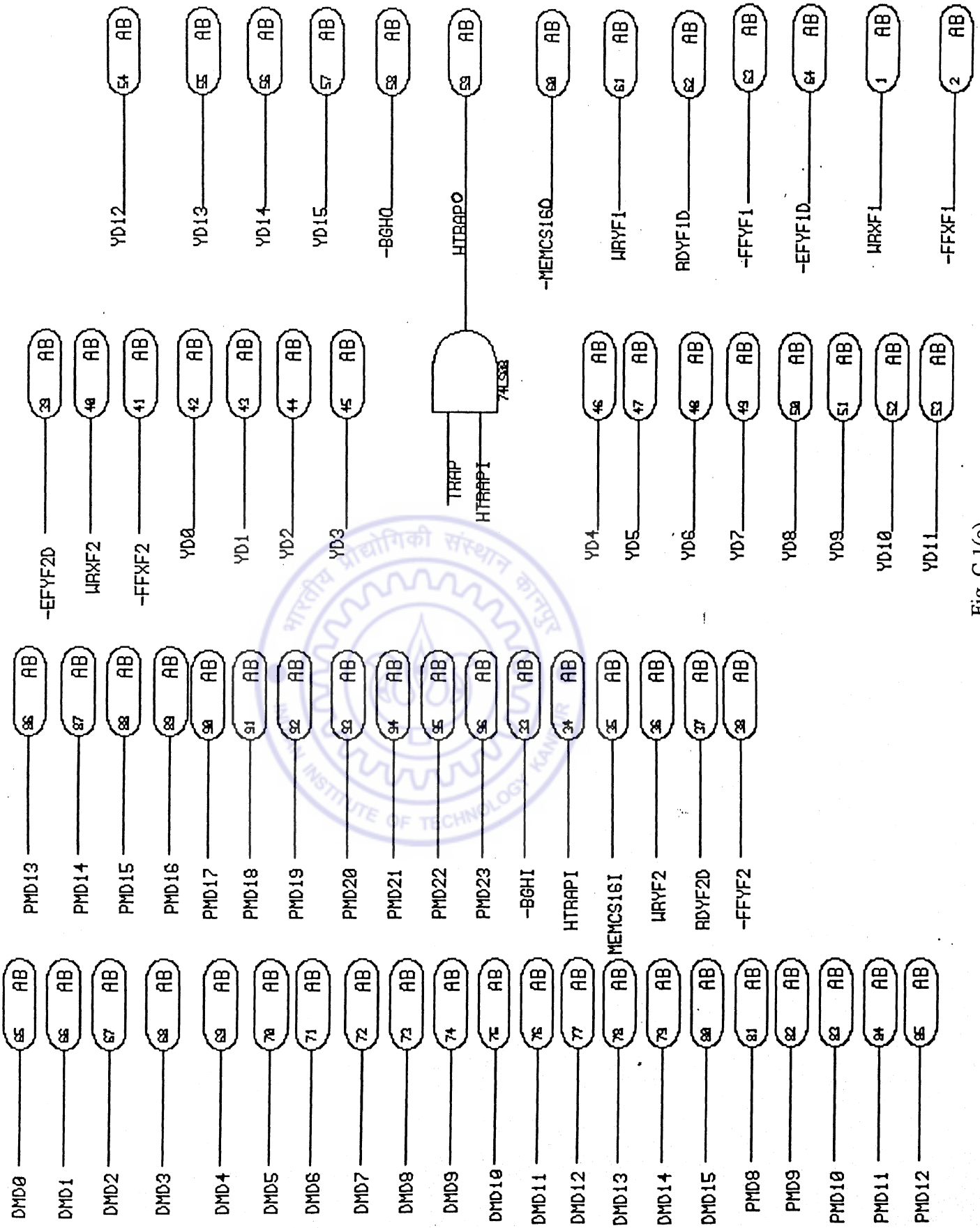
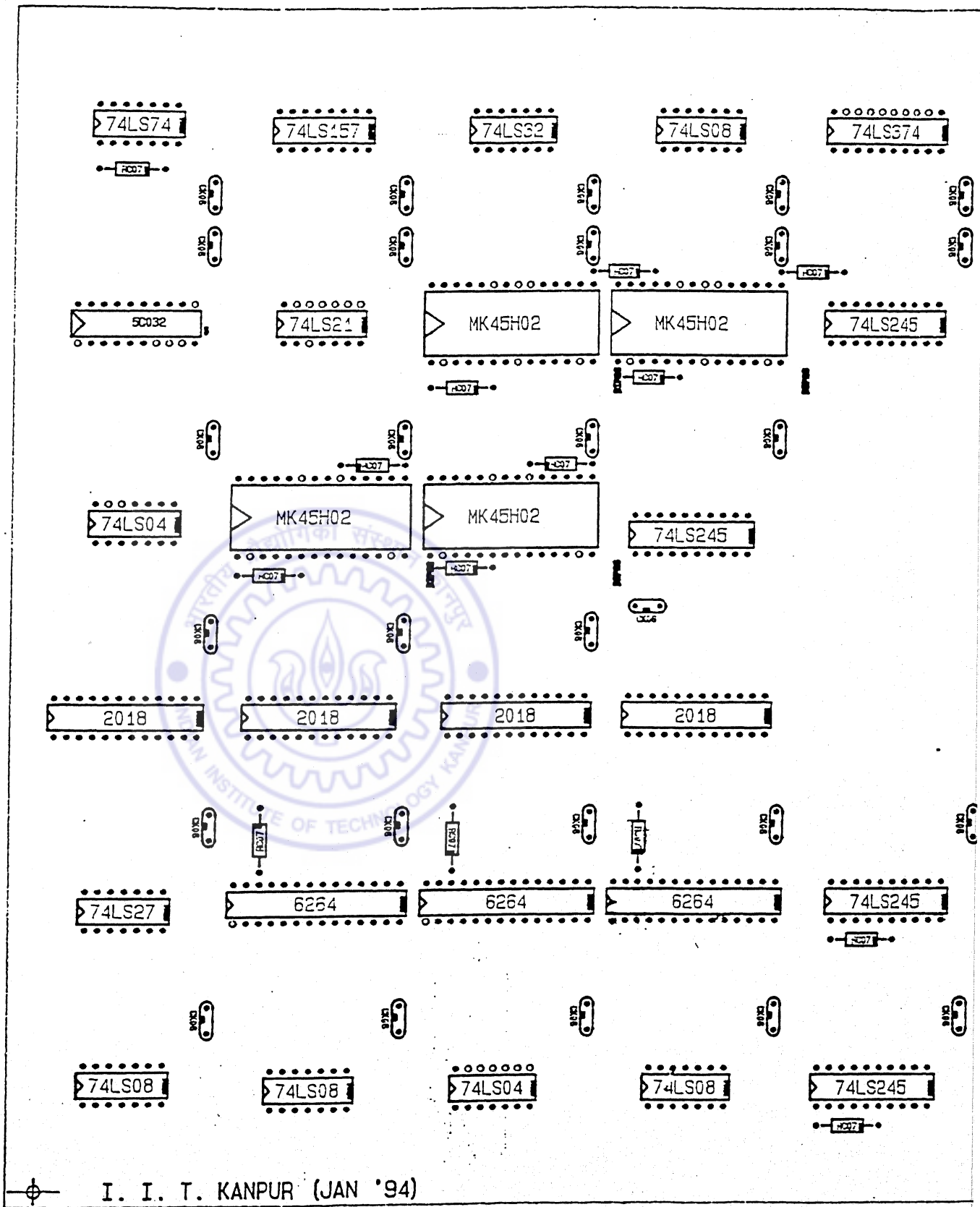
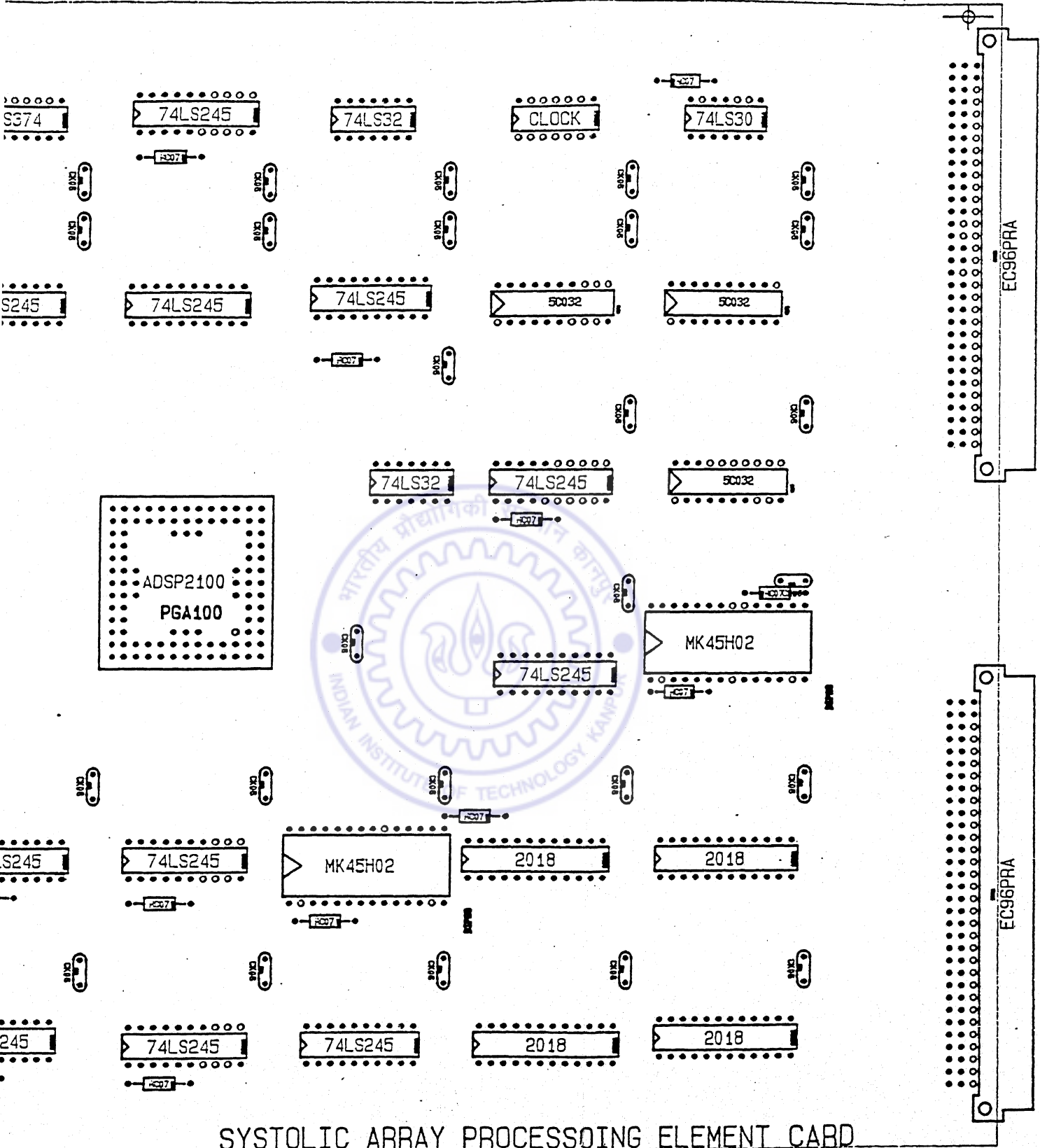


Fig. C.1(o)

APPENDIX D





SYSTOLIC ARRAY PROCESSING ELEMENT CARD

Fig. D.1