The Development of a Multi-layer ~~System~~ architecture for Image Processing

Yu Fai FUNG

Image Processing Group
Department of Physics and Astronomy
University College London

Thesis presented for the Degree of

Doctor of Philosophy
in the
University of London
1991

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

ABSTRACT

The extraction of useful information from an image involves a series of operations, which can be functionally divided into low-level, intermediate-level and high-level processing. Because different amounts of computing power may be demanded by each level, a system which can simultaneously carry out operations at different levels is desirable. A multi-layer system which embodies both functional and spatial parallelism is envisioned. This thesis describes the development of a three-layer architecture which is designed to tackle vision problems embodying operations in each processing level.

A survey of various multi-layer and multi-processor systems is carried out and a set of guidelines for the design of a multi-layer image processing system is established. The linear array is proposed as a possible basis for multi-layer systems and a significant part of the thesis is concerned with a study of this structure. The CLIP7A system, which is a linear array with 256 processing elements, is examined in depth. The CLIP7A system operates under SIMD control, enhanced by local autonomy. In order to examine the possible benefits of this arrangement, image processing algorithms which exploit the autonomous functions are implemented. Additionally, the structural properties of linear arrays are also studied.

Information regarding typical computing requirements in each layer and the communication networks between elements in different layers is obtained by applying the CLIP7A system to solve an integrated vision problem. From the results obtained, a three layer architecture is proposed. The system has 256, 16 and 4 processing elements in the low, intermediate and high level layer respectively. The processing elements will employ a 16-bit microprocessor as the computing unit, which is selected from off-the-shelf components. Communication between elements in consecutive layers is via two different networks, which are designed so that efficient data transfer is achieved. Additionally, the networks enable the system to maintain fault tolerance and to permit expansion in the second and third layers.

Contents

List of Figures

List of Tables

Acknowledgement

I wish to thank my family, especially my parents, for their love and support. I should like to thank my supervisor, Dr. T.J. Fountain, for his encouragement, guidance and patience. Thanks are also due to members of the Image Processing Group for their help and advice in all aspects.

Finally, I should like to thank Mr. D.J. Chandler for proof-reading and correcting the text of this thesis.

Chapter 1
Introduction

1.1 Introduction

Image processing can be applied to a wide range of problems including medical diagnosis, remote sensing, robotics and industrial inspection. The ultimate goal of image processing is the extraction of useful information from a picture which can then provide a solution to the target problem. In industrial inspection, for example, image processing can be used to examine components coming from a conveyor belt with the result that defective objects can be identified and discarded automatically. A defective object can be identified by measuring the physical parameters of objects coming from the conveyor belt and comparing them with those of a model object. If the difference between the measured values and the model data exceed a pre-defined threshold level then the object is regarded as defective. In automatic vehicle guidance, the image to be processed may include different objects, such as other vehicles, traffic lights and pedestrians with the movement of the automatic vehicle dependent on the spatial relationships between them. In such a situation, the processing operations may include first the identification of the objects and then an evaluation of their spatial relationships. The two examples given demonstrate the varying processing requirements imposed by different problems.

A digital image is organised as a 2-D array of picture elements known as pixels, the value of each representing the image intensity level at that particular location. The size of an image and the number of bits used per pixel vary according to the application. A LANDSAT (Land Satellite) image [1], for example, is represented by a 2340x3380 array of pixels, with 7-bit per pixel. A single frame of LANDSAT imagery consists of a set of 4 images taken at different spectral windows. In Chinese character recognition [2] on the other hand, the size of a character image is 64x64 pixels and each pixel is represented by a 8-bit integer.

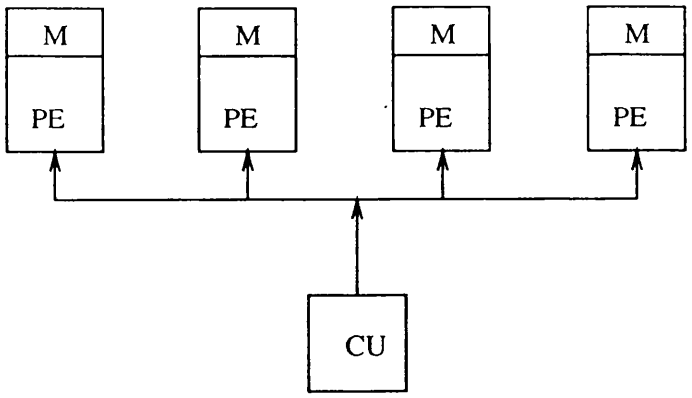
If an image has a resolution of 256x256 pixels, with each pixel carrying 8-bit of information, then a total of 64 Kbytes of data is constructed. In many applications, for example a traffic monitoring system [3], images are processed continuously at up to 25 images per second. This highlights the amount of data involved in some computer vision problems. An image operation must apply to the entire image (i.e. every single pixel), meaning that for a 256x256 pixels image the operation has to be repeated 65536 times. A thresholding operation will take 100ms to complete if applied to a 256x256 pixels image, using the SUN3 computer running at 16MHz, for example. A

processing task consists of a sequence of operations, with the processing time proportional to their complexity. Because of the amount of data and the processing operations involved, special computing systems embodying different degrees of parallelism have built or proposed since the early 1960s.

Parallel architecture can provide the solution to the enormous computing power demanded by image processing applications and can be classified according to the parallelism employed for the instruction stream and data stream [4]. There are three distinct classes: Single Instruction Multiple Data (SIMD); Multiple Instruction Single Data (MISD); and Multiple Instruction Multiple Data (MIMD). These various control mechanisms can be illustrated graphically as shown in Fig. 1.1. A SIMD machine consists of a single controller issuing instructions to all processing units in the system, which then operate on their own data stored in the processors' memory. In a MISD system, individual processing elements include a controller, but share a common memory unit. Each processing element operates on the same data set but perform different operations. The processing element of a MIMD system consists of both a memory unit and a control unit. Individual elements can process different data and perform various operations. There are systems which combine both SIMD and MIMD properties, for example, the PASM [5]. A survey of the various parallel systems is found in [6].

The pipeline architecture (see Fig. 1.2) can be classified as a MISD system, but is always regarded as a special category [7]. A pipeline system consists of a set of processing elements connected in sequence so that data entering the pipeline at the first processing element will leave at the last. The output from one processing element is passed directly to the next element in the pipe. Each element can perform different operations on the data and, if there is a continuous input of data, at one stage all processing elements in the pipeline will be active and the system be in MIMD mode.

How can parallelism be applied to an image processing system? As a sequence of operations is applied to image data to extract information, parallelism can be achieved in two different ways: spatially and functionally. Spatial parallelism refers to a system's capability to operate on a group of data in a single operation. A digital image is arranged as a 2-D array of pixels, as stated earlier, and many operations have to apply to every pixel in the image. An example of spatial parallelism is the partition of the given image into regions, each of which is processed by an individual processor. The partition can be fine grain with each pixel processed by a single processing element (PE) and machines like the CLIP4[8], DAP[9], MPP[10] are examples of this class. The BASE[11] system, on the other hand, employs a coarse grain architecture

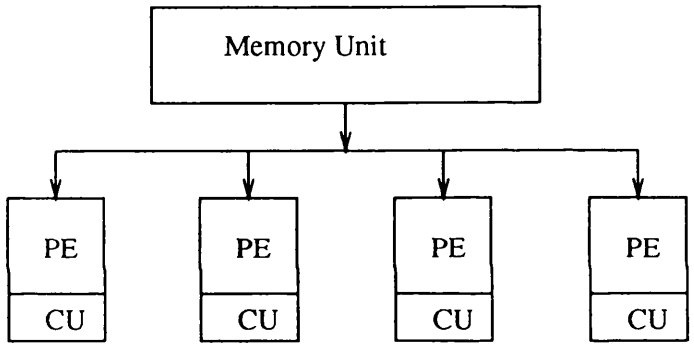


M --- Memory

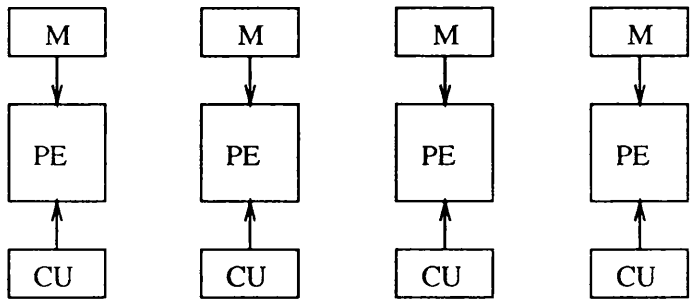
PE -- Processing Element

CU -- Control Unit

(a) SIMD controlling mechanism

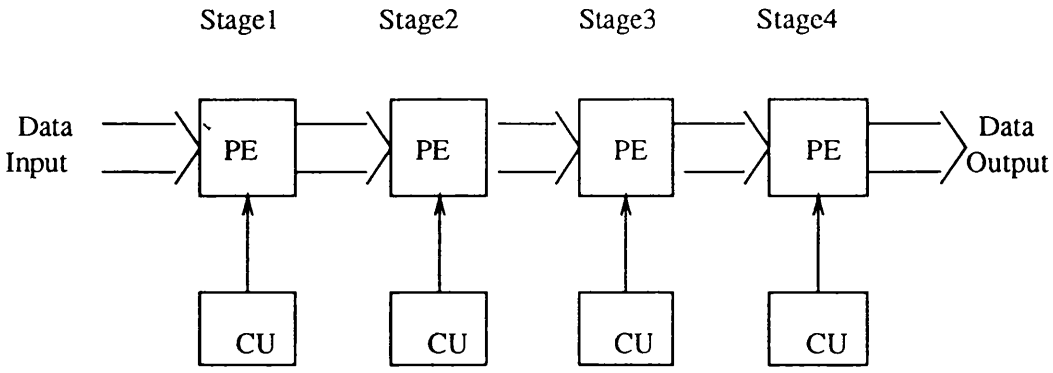


(b) MISD controlling mechanism



(c) MIMD controlling mechanism

Figure 1.1 Flynn Taxonomy



PE -- Processing Element

CU -- Control Unit

Figure 1.2 A pipeline architecture

with only 64 PEs, each assigned to a region of 32x32 pixels. Because each processor performs an identical function on a unique set of data the SIMD control mechanism maps well onto such an organisation. Generally speaking, spatial parallelism employs a large number of processing elements and can be very effective in performing pixel to pixel transformations.

Functional parallelism involves partitioning the complete process into sub-processes and assigning different processing elements to perform various functions. In a functionally parallelised system each PE operates on the entire image data and the result from one is passed to the next for further processing. Functional parallelism maps well onto the pipeline architecture: the FLIP[12] is an example in this category (other pipeline systems are included in surveys presented in [6] and [7]), with its 16 special processors performing different functions. Each processing element is assigned a particular operation on the entire data set and overall system performance is improved by specific tailoring of the processors to the operations.

The above analysis suggests that a system's performance could be maximised by exploiting both spatial and functional parallelism and a multi-layer system is thus envisioned. A layer in such a system would embody several processing elements, each operating on its own set of data, with output generated by one layer being passed to the next for further processing. In relation to such a system, two obvious questions are raised: the first concerns the number of layers and the second the number of elements included at each layer. Answers can be obtained by further examination of the image processing problem. Image processing involves a sequence of operations that can be classified into three levels [13], [14]: low, intermediate, and high. There are, however, many other classifications [6], [15] suggesting that the problem could be divided into iconic and symbolic domain only. Low-level processing usually means image to image transformation and may involve algorithms for noise removal and simple feature extraction such as edge detection. Intermediate-level processing operates on symbols extracted from the input image and generates a working description of features represented by them. High-level processing works on features, usually involving some real-world models, and derives a solution for the problem. This taxonomy is far from complete, however, many vision problems share such a structure (as described in Chapter 6). It is important to point out that not every vision problem involves processing at all levels but based on this problem structure, a multi-layer system would consist of three layers and the number of PEs employed by each layer should be proportional to the quantity of data being processed. After introducing the infrastructure of the envisioned multi-layer system, the following chapters will

describe the development of such a system.

The development of the multi-layer system begins with a survey of existing systems. The survey presented in Chapter 2 concentrates on the hardware features of such systems and includes a look at the structure of the processing elements; the communication network between elements and between layers and the controlling mechanism employed. The survey will show that existing systems employ the mesh-connected array as their basic structure, which maps well to the initial data type - image. However, the data types evolved in intermediate and high level processing (discussed in Chapter 6) may not be effectively represented by a 2-D structure and this leads to the proposal of using a linear array as an alternative to a 2-D array for the construction of a multi-layer system.

A linear array is cheaper and easier to construct. More importantly it can be adapted to represent data structures evolved at all three levels. Generally speaking, a linear array with N PEs is employed to process an image with N^2 pixels. Because the number of processing elements is significantly less than an N^2 PE mesh-connected array, the performance of a linear array may be affected. Local autonomy provides one possible mechanism to improve performance. In Chapter 3 different aspects of local autonomy are discussed and a survey of existing or proposed linear arrays is included. The survey attempts to establish a better understanding of linear arrays.

The linear array is chosen as the basic unit for a multi-layer system but there are parameters related to its design which are still undefined. This necessitates an investigation of the following areas:

1. The effectiveness of local autonomy;
2. The structural properties of a linear array;
3. Processing requirements at different levels of operations; and
4. The relationships between the structure of the problem and the architecture of the system.

In order to carry out the investigation a specific linear array is selected. The one which is chosen is the CLIP7A system, a detailed description of which is included in Chapter 4. An understanding of both hardware and software aspects of the CLIP7A system will assist the reader's comprehension of the algorithms described in chapters 5 and 6.

Chapter 5 describes the algorithms implemented in the CLIP7A system in order to investigate the structural properties of a linear array and the effectiveness of local autonomy. The results obtained provide information useful for the design of individual layers of the multi-layer system. In addition, information concerning the design

of the overall structure of the multi-layer system, and the computing requirements at each layer, is obtained by applying the CLIP7A system to an integrated vision problem. The algorithms devised for solving the problem are described in Chapter 6. The modified DARPA benchmark [63], which embodies processing at three levels, is chosen as an example of a scene analysis problem.

Utilising the results obtained from chapters 5 and 6, the design of a three-layer system is conceived. Chapter 7 describes the overall structure of the system and concentrates on the architecture of the inter-layer communication networks. Chapter 8 is concerned with the design of four major system components. These include the processing elements; the implementation of the inter-layer communication network; the controlling mechanism and the system software.

Finally in Chapter 9, a brief summary of this work is given, along with discussion relating to the cost-effectiveness of the proposed multi-layer system.

Chapter 2
Multi-layer Image Processing Systems

2.1 Introduction

Image processing is a computational intensive problem because of the large quantity of data and operations involved. Parallelism provides one solution to this problem and different parallel systems (for example the mesh-connected processor arrays mentioned in the last chapter) have been built for image processing. A parallel system capable of processing a huge amount of data and of performing different operations at the same instant is desirable. A multi-layer system, embodying both spatial and functional parallelism, satisfies both requirements. Spatial parallelism refers to the system's ability to process multiple data at the same instant, whilst functional parallelism describes the system's ability to perform different functions simultaneously.

The development of a multi-layer image processing system must begin with a survey of existing multi-layer systems. A pyramid system is a typical multi-layer architecture and six different systems are considered in the following sections. The survey examines first, the structures of the various processing elements employed by these systems; second, the connectivity between elements in the same layer and in consecutive layers and, finally their controlling mechanisms. The objective of the survey is to assess the suitability of the pyramid structure for the envisioned system. Three different multi-processor systems are also included in the survey, each representing a special category of parallel architecture and their inclusion provides additional information relevant to the development of the multi-layer system.

2.2 Pyramid Systems

A pyramid system, see Fig. 2.1, is a multi-layer architecture employing 2-D array as its basic building unit. The base layer (layer 0) of a system usually has a size equivalent to the input image, with the next layer having a reduced number of PEs. The system converges until the apex, consisting of a single element, is reached. The flow of information usually starts from the base layer and advances towards the apex. In most systems, a constant convergence ratio (the number of PEs in layer (N-1) divided by those in layer (N)) is used. Elements in a pyramid system are connected both horizontally, within a layer, and vertically, between consecutive layers. For vertical connectivity, the terms father and son are used to describe the physical relationship between the elements. When two elements in two consecutive layers are connected,

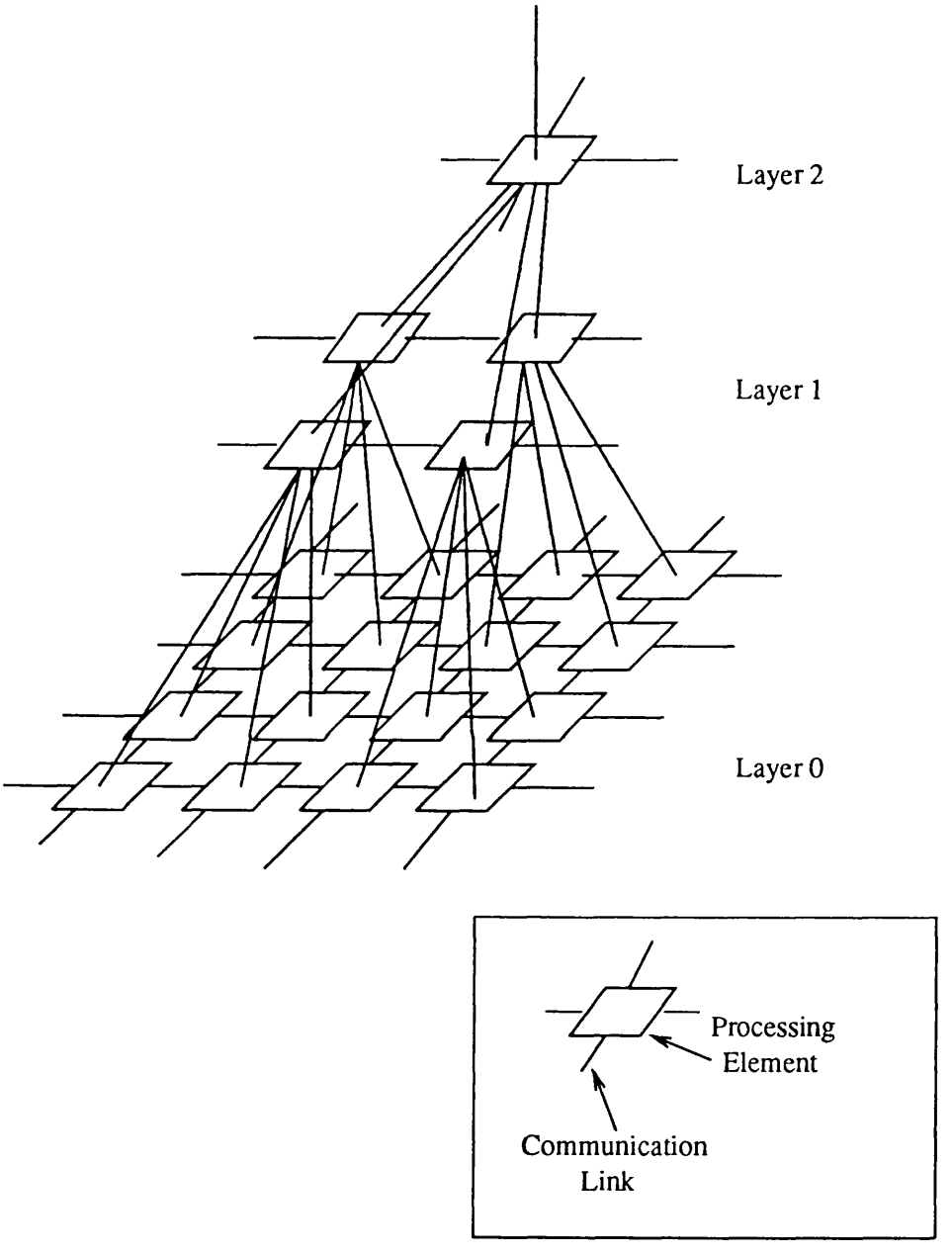


Figure 2.1 A pyramid architecture

then the element in the upper layer is called the father and the element in the lower layer the son. The son to father ratio is equal to the convergence ratio.

Pyramid systems can be divided into two different categories: homogeneous and heterogeneous. A homogeneous pyramid incorporates an identical PE in all its layers which is, usually, a simple processor. Heterogeneous pyramids, on the other hand, employ different types of processing elements in different layers with more powerful processors generally used in the upper layers.

2.2.1 Homogeneous Pyramids

Four homogeneous pyramid systems are included in this survey. According to their inter-layer connectivity, these systems can be sub-divided into two classes: quad-pyramid and binary-pyramid. In a quad-pyramid every PE is connected to 4 sons (excepting those in the base layer), whilst only 2 sons are provided in a binary pyramid. Convergence ratios for quad-, and binary, pyramids are 4, and 2, respectively. Homogeneous systems include those described below.

GAM Pyramid

The GAM pyramid [16] is a 5 level quad-pyramid developed by the George Mason University. It has a base level of 16x16 PEs and is devised to process 16x16 pixel binary images. Each PE is connected to 4 siblings. The PEs employed, as shown in Fig. 2.2, are those used in the Massively Parallel Processor (MPP) [10] developed by the Goodyear Corporation. Each MPP PE includes a set of 6 1-bit registers, a 30-bit shift register, a full adder and multiplexed neighbour inputs (4-connected). Each MPP chip embodies 8 PEs and runs in 10MHz. The MPP pyramid unit is enhanced by 8 Kbits of RAM and a geometric unit that provides paths for data transfer between PEs either in the same layer or in the layers above and below. The function of the geometric unit is to duplicate or select data when it is being transferred between layers due to the number of PEs at each layer being different. Data is duplicated when it is moved from a higher to a lower level, whilst sampling (or selecting) is required for data passing from a lower to a higher level. There is a feedback output, provided by the MPP chip, that indicates the data bus state of each PE. The SUM-OR of all the active PEs is available at each level. The state of the mask register, included in each PE, can disable or enable the activity of that particular PE so that conditional operations can be achieved.

The special feature of the GAM system is the ADDER pyramid receiving its input from the level 4 array and generating a 7-bit output from its 64 inputs (i.e. the

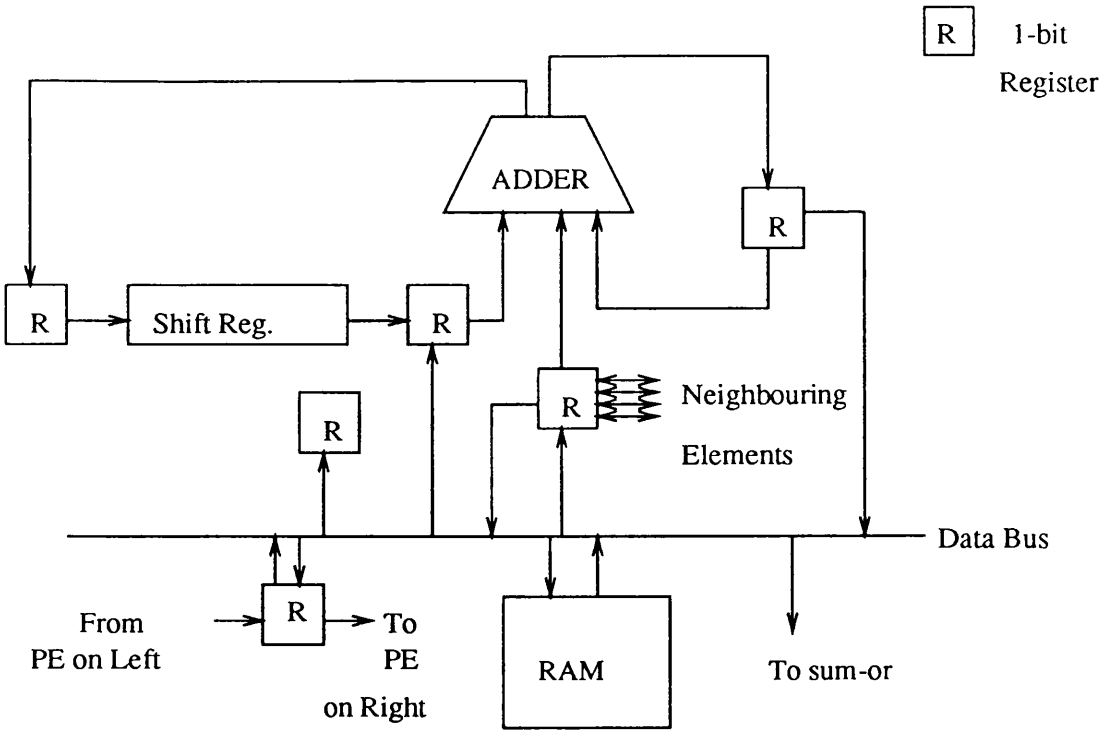


Figure 2.2 Block diagram of a MPP PE

number of pixels with the value of '1' on the level 4 array). The ADDER pyramid provides a speed up for counting the number of object pixels by a factor of 23 comparing with using the adder in the MPP alone. The GAM pyramid is under SIMD control except in data transfer between layers. During data transfer one layer is specified as the sending agent, whilst another is the receiving agent. The system is programmed using the GAM pyramid assembly language, which is a combination of micro-code for the pyramid processing elements and control unit commands.

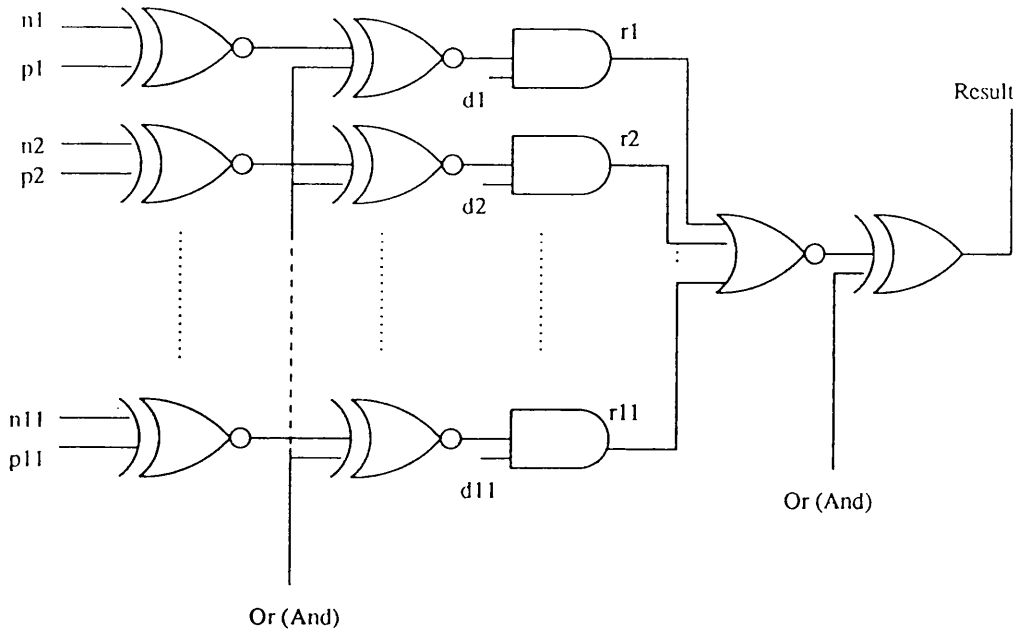
Hierarchical Cellular Logic (HCL) Pyramid

The HCL [17] pyramid is constructed at the University of Washington. The basic building unit of the pyramid is the HCL chip. The HCL chip is a custom VLSI integrated circuit fabricated in NMOS technology and a single chip, a 64 pin device, embodies a 4x4 array of PEs. The system consists of 4 levels and has a base layer of 8x8 PEs. The HCL system is a quad-pyramid and each PE is connected to its 8 nearest neighbours in its own layer.

The PEs of the HCL pyramid are very simple and are designed to perform two principal operations: AND_Match and OR_Match. The circuitry for the AND_Match/OR_Match logic is depicted in Fig. 2.3. Both operations compare a specified pattern with a PE's neighbouring elements, i.e. 13 PEs in total. In the case of the AND_Match, a '1' is written to a PE's accumulator only if the pattern matches with all neighbouring elements, otherwise a '0' is put to the accumulator. The OR_Match operation will generate a '1' if the pattern matches with any of the neighbouring elements and if not a '0' is written to the accumulator. All operations performed by the pyramid are a combination of these operations. Each PE consists of 3 1-bit registers used for temporary data storage and an extra 8 Kbits of external memory is available. The external memory of the base layer is accessible by the host thus providing the communication media to the outside world. The control of the pyramid is SIMD and the host is an IBM PC.

Pyramid Architecture for Parallel Image Analysis (PAPIA)

The PAPIA system [18] is being developed in Italy by a consortium of universities and industrial companies. It comprises four units, namely the Global Control and Scalar Processing Unit, the Pyramid Control Unit, the Multiprocessor Pyramid Unit (MPU) and the Active Memory Unit. Only the Multiprocessor Pyramid Unit will be discussed in this section.



n represents neighbourhood input

p and d specify the control pattern

Figure 2.3 AND_Match/OR_Match circuit of the HCL pyramid

The MPU is made of 8 planes and arranged as a quad-pyramid, whilst each PE is connected to its 8 nearest neighbours. The PE, as shown in Fig. 2.4, consists of a 1-bit ALU, 256 bits of local memory, near neighbour selection unit, an I/O port and a registers and Boolean operators unit. The PE is implemented as a VLSI custom-designed circuit which is fabricated in 4μm Si-gate NMOS technology. Five PEs are integrated into a single chip and are arranged as one father and 4 sons. A PE's near neighbours are divided into two groups: horizontal and vertical. A PE's siblings belong to the horizontal whilst its father and sons are vertical. A PE can only communicate with a single group at each instance and the near neighbour selection unit will choose the proper group. Two shift registers are included in the registers and Boolean operators unit and are used to store temporary data.

Local control is achieved by disabling the activity of a PE according to the value of a mask register. Two masking levels are possible: global and local. When the global masking is set then PEs in the same plane will be disabled, while local masking is effective only when the PE is globally enabled. A feedback signal reflecting the current status of the ALU is provided by each PE. The global state of a plane is available by summing all feedback from the PEs by means of an OR gate.

Two clock frequencies are employed; they are for computation and for data transfer inside, or outside, the pyramid. The clock frequency used for data transfer is faster than the frequency used for computation and is dependent on the speed of the image input/output device. The two clocks overlap so that processing and image input/output can be performed concurrently.

The system can be run in either SIMD, or multi-SIMD, mode. In multi-SIMD mode the pyramid is divided into three sections and planes in each section are under SIMD control of an independent controller.

SPHINX

The SPHINX [19], [20] is being developed by the Paris Sud University and the ETCA Defence research lab of France. It is regarded as a binary pyramid but a different inter-layer connectivity is employed and each PE is alternately the father of 2 sons of the same X and Y coordinates. The different inter-layer connectivity means that the system consists of 15 layers with a base of 128x128 PEs and each PE is connected to 4 siblings.

Each PE (see Fig. 2.5) consists of a bit-serial ALU, a 128x1 bit dual-ported memory and some special purpose 1-bit registers which are devised for communication with other PEs, or for the activity control of the PE.

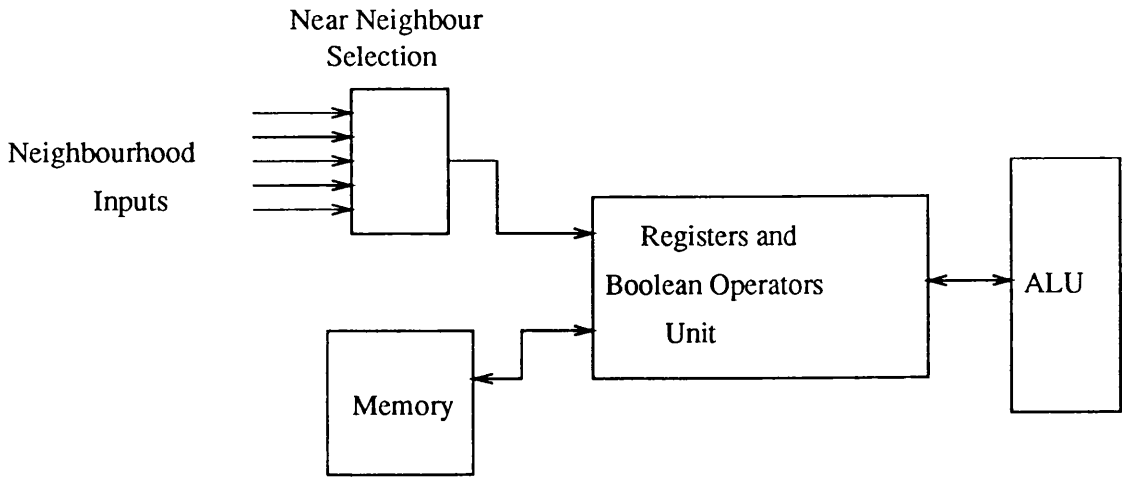


Figure 2.4 Block diagram of the PAPIA PE

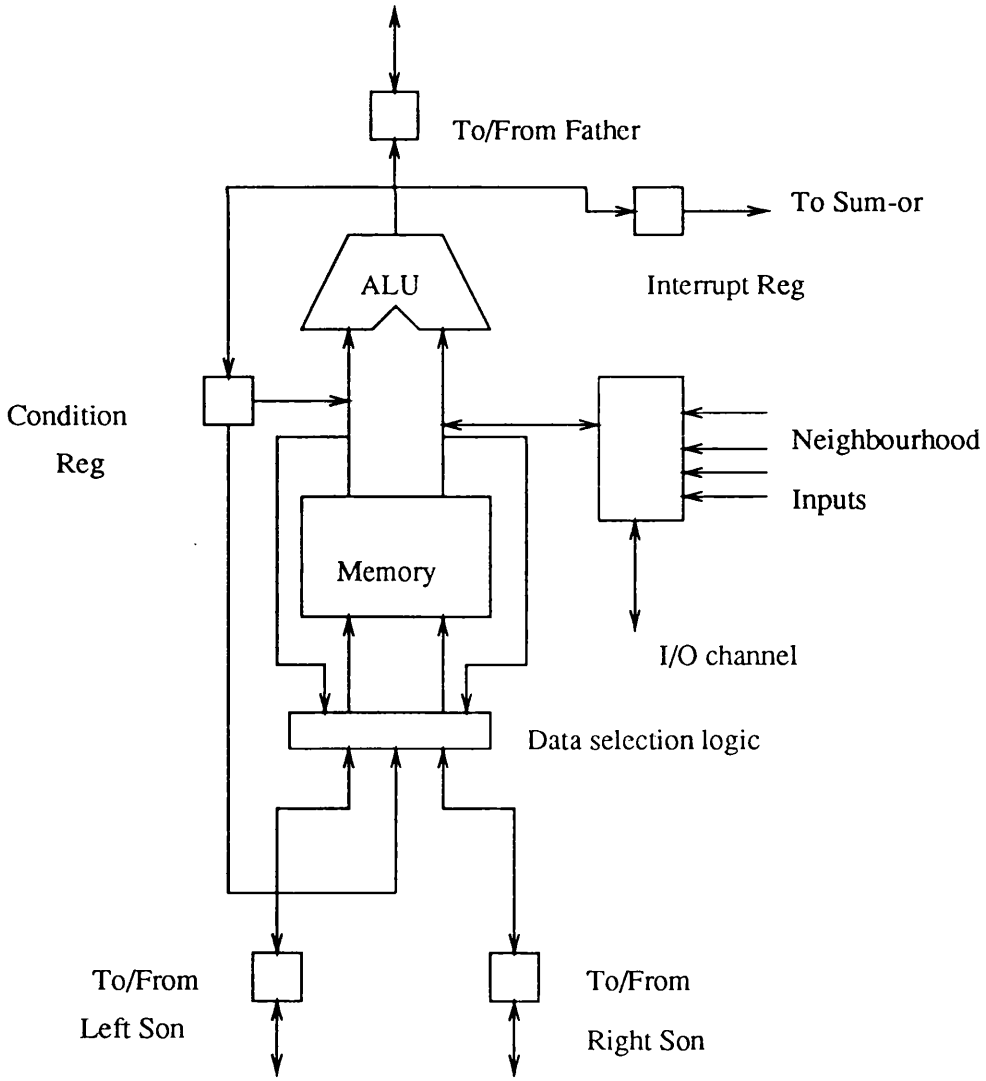


Figure 2.5 Block diagram of the SPHINX PE

The ALU is capable of performing all the 16 logical operations on its two inputs and four arithmetic functions (ADD, SUB, NADD and NSUB) with the carry flag generated accordingly. There are 128 bits of internal memory which can be accessed by two independent busses. This allows two read/write operations to be performed simultaneously. There is a bit-serial I/O channel enabling a PE to access external memory. The register set includes a condition register, a carry register and an interrupt register. The condition register is used to inhibit the operation of a PE, according to its content. This masking capability enables the implementation of IF...THEN...ELSE control structures. The carry register is used either as a source or destination for arithmetical operations. The interrupt register is used to centralise a Boolean information on a single layer.

Communication between PEs in different layers is through three registers, one for the father and two for the sons. Reading from, and writing to, those registers is controlled so that data passing between layers is unidirectional. Data transfer between neighbouring PEs in the same layer is through a 4 direction shift register.

The control of the pyramid is multi-SIMD. PEs in the same layer perform the same operation but independent routines can take place concurrently on different layers. The controlling unit in each layer embodies two controllers (high level and low level) which are linked by a FIFO. The dual controller structure is devised to achieve inter-layer synchronisation. The high level controller issues instructions to be performed by the layer to the FIFO. The instructions will not be executed until a specific condition between the layers is reached. The function of the low-level controller is to monitor the status of a layer and send instructions to that layer only when the condition is reached.

2.2.2 Heterogeneous Pyramids

Heterogeneous pyramids employ different processing elements in different layers. In most cases, excluding the host, there are only three layers and the convergence ratio between consecutive layers is high. Each layer is dedicated to one level of image processing tasks and processing power increases with the level of the pyramid. A description of heterogeneous systems follows.

The Image Understanding Architecture (IUA)

The IUA [21], [22] is being developed in the University of Massachusetts and is designed to function as an automatic scene interpreter, with a three level structure envisaged. Starting from the bottom, the layers are the Content Addressable Array

Parallel Processor (CAAPP), the Intermediate Communications Associative Processor (ICAP), and the Symbolic Processing Array (SPA), as shown in Fig. 2.6.

The CAAPP is a 512x512 PE mesh-connected array performing image to image transformations. Each PE includes 5 one-bit registers, an ALU, data routing circuitry and 320 bits of RAM. It is implemented as a custom-designed circuit incorporating 64 PEs in a single chip. The CAAPP is supported by two special features, global summary feedback and the Coterie network. The global summary mechanism is an array-wide logical 'OR' output indicating whether any CAAPP cells are in a given state. A counting operation which reports the number of cells in a given state is provided. The Coterie network is a switch network enabling processors having similar properties to be grouped together. Processors in a coterie communicate through a bit-wide bus so that groups of processors can execute globally broadcast instructions using locally broadcast data.

The ICAP is a square grid array of 64x64 PEs and will carry out symbolic processing. Each PE consists of a CPU, 256 Kbytes of local RAM, 384 Kbytes of dual-ported memory and network communication hardware. The CPU is a Texas Instruments TMS320C25 16-bit digital signal processor. The function of the dual-ported memory is for communication between the ICAP and the CAAPP, and the SPA. Like the CAAPP, the ICAP provides a global 'OR' summary mechanism reflecting the status of the layer. In the prototype system with only 64 PEs, intra-layer communication is via a bit-serial crossbar switch network.

The SPA is devised for knowledge-based processing and will consist of 64 powerful processors (the Motorola MC68020 or similar processors) capable of implementing the LISP language. The detailed architecture of the SPA has not yet been fully defined. A full-connection network is suggested for intra-layer communication. Top-down control of the ICAP by the SPA is also proposed.

The convergence ratio between layers is 64, i.e. each SPA communicating with 64 ICAPs and each ICAP linked to 64 CAAPPs. Communication between layers is through the dual-ported memory of the ICAP. The 384 Kbytes of dual-ported memory is divided into a 256 Kbyte block and a 128 Kbyte block. The 128 Kbyte block is used to communicate with the SPA and the 256 Kbytes memory serves as an interface with 64 CAAPP PEs, that is 32 Kbits per PE.

The controlling mechanism is different at each level. The SPA processors operate in MIMD whilst the ICAP can operate either in synchronous-MIMD mode or pure MIMD mode. In synchronous-MIMD mode the ICAP processors execute the same programme codes, but have their own instruction pointers so that they can

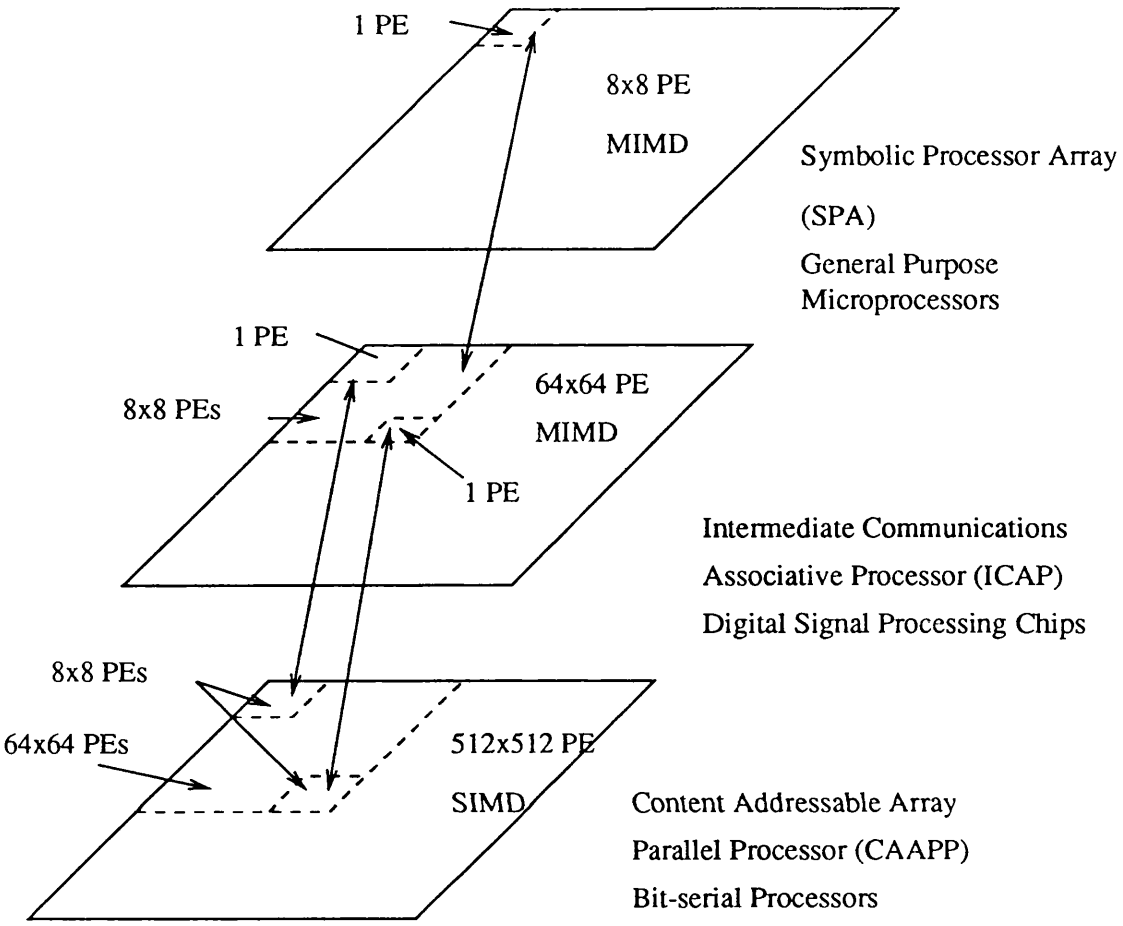


Figure 2.6 An overview of the IUA system

branch independently, and globally synchronised at each stage of processing. The CAAPP is under SIMD control of the array control unit.

Warwick Pyramid Machine (WPM)

The WPM [23], [24] is a 4 layer system orientated towards image based problems, such as image understanding and computer graphics. The structure of the pyramid machine is divided into a SIMD array, a controller array, a MIMD array and the host, as shown in Fig. 2.7. The convergence ratio between the SIMD array and the controller array is 256, whilst a one to one ratio is used between the controller array and the MIMD array.

The SIMD array is mesh-connected and consists of 128x128 PEs for performing image to image transformations. These are the AMT Distributed Array Processors (DAP). The DAP is a bit-serial processor incorporating a 1-bit ALU and four 1-bit registers. The simplicity of the PE means that 64 PEs are integrated into a single chip. Data storage for the processors is provided by external memory and there are 32 Kbits per processor.

There are 64 elements, arranged in a 8x8 array, in the controller layer. The major function of the PE is to issue instruction streams to the SIMD array, therefore each PE consists of a micro-instruction sequencer (an AMD29331). Other components utilised in the PE include a micro-programmable 16-bit processor (AMD29116), 16K x 68bit micro-code store and 2 Kwords dual-ported memory. The dual-ported memory is used as a communication channel between the controller and the processor in the MIMD array.

The MIMD layer is a 8x8 PE transputer array for symbolic processing. The transputer used is the T414 32-bit processor, each associates with 256 Kbytes of external memory. Each transputer is connected to its four nearest neighbours and communicates with a single controller in the controller array.

The host is used as an user interface and also provides mass storage for the system. Software facilities, such as text editor and programming language compiler, are supported.

Control of the system is provided by either the controller array or the MIMD array. The SIMD array is divided into 64 16x16 sub-arrays, each of which under the control of a single element in the controller array. Such an arrangement, i.e. 16x16 SIMD PEs and 1 controller, is called a cluster. Each cluster can operate independently of other clusters resulting in a multi-SIMD functioning mode. Instructions come from

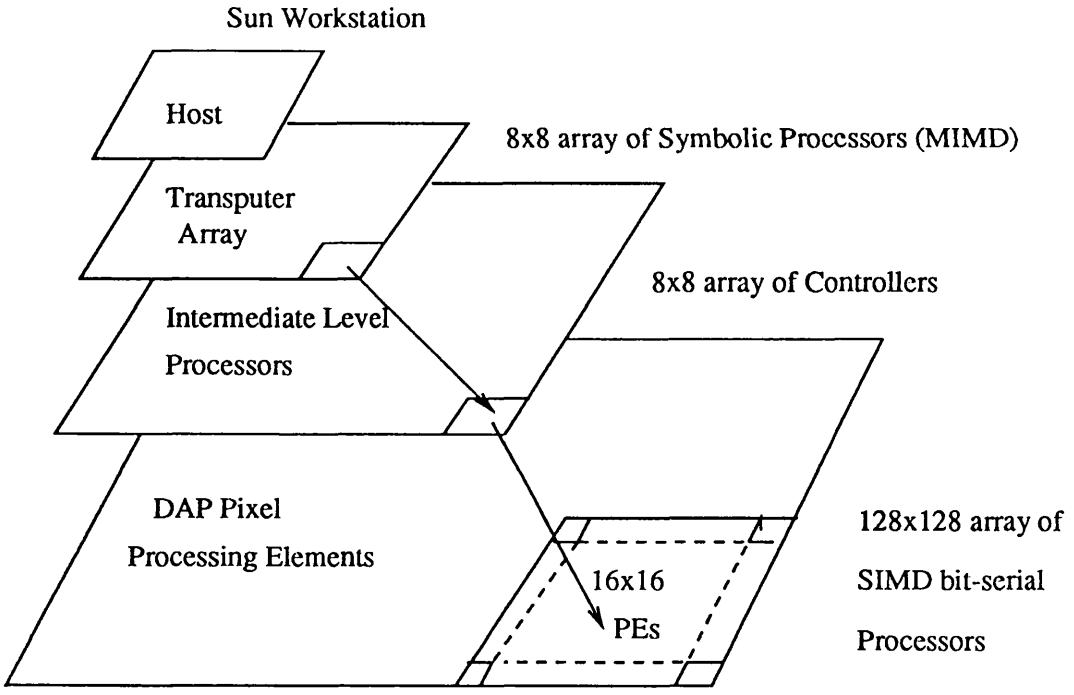


Figure 2.7 An overview of the Warwick Pyramid Machine

the micro-code store in the controller and are broadcast to all PEs within a cluster. Elements in the MIMD layer can also direct the operations of a cluster by calling up an instruction through a software protocol using the dual-ported memory in a controller.

Communication between a controller and the SIMD PEs in the cluster is through a special register mapped into the register space of the controller. Additionally, a counting mechanism, reporting the number of SIMD processing elements of a given status, is provided.

Programming of the system is through two programming languages, the Parallel C and cluster assembly language. The Parallel C is used to programme the transputer array. The cluster assembly language is for the programming of both the controller and the SIMD PEs in a cluster.

2.3 Other Multi-processor Systems

Table 2.1 summarises the properties and features of the machines described in the previous section. In addition to the above pyramid systems, there are three systems, namely the Connection Machine [25], [26], the PIPE [27], and the PUMPS [28], which are included in the following discussion. Each system represents a class of multi-processor architecture. Useful information relevant to the development of a multi-layer system will be extracted from these architectures. The discussion will concentrate on the important features of each system and details of the processing elements will not be included.

Connection Machine

The Connection Machine is a massive data parallel computing system developed by the Thinking Machine Corporation. The maximum number of PEs incorporated in the system is 65536 and each PE is connected to its four nearest neighbours. Each PE, as shown in Fig. 2.8, consists of a single bit ALU, 64 Kbits of memory, four 1-bit registers and communication interface logic. 16 PEs, with the exception of their memory, are integrated into a single chip. The ALU can perform two Boolean functions with its three inputs and arithmetic operations are performed as a combination of the Boolean functions. The performance of the PEs can be enhanced by a floating-point accelerator. There are two possible options for this accelerator, single precision (32-bit) or double precision (64-bit). The accelerator is also implemented as a special purpose VLSI chip and each accelerator supports the operations of 32 PEs.

System	GAM	HCL	PAPIA	SPHINX	IUA	WPM
Type	Ho quad	Ho quad	Ho quad	Ho binary	He	He
Number of layers	5	4	8	15	3	4
Size of base layer	16x16	8x8	128x128	128x128	512x512	128x128
Layer ratio	1:4	1:4	1:4	1:2	1:64	1:256 1:1
ALU	1-bit	Special functions	1-bit	1-bit	1-bit 16-bit 32-bit	1-bit 16-bit 32-bit
Control Mode	SIMD	SIMD	Multi-SIMD	Multi-SIMD	SIMD and MIMD	Multi-SIMD and MIMD

Ho == Homogeneous and He == Heterogeneous

Table 2.1 Parameters of different pyramid systems

The most important feature of the Connection Machine is its inter-element communication mechanisms including both a router network and the NEWS grid. The router network enables a processor to communicate with any other processor in the system. There is one router per 16 PEs which is incorporated in the processor chip. The routers form a 12 dimensional hypercube network [29] meaning that each router is connected to 12 other routers. Each router is given a unique number and two routers, e.g. i and j, are connected if and only if $|i - j| = 2^k$ for some integer k. A message originating from a PE may pass through a series of routers before reaching its destination. To achieve this, each router embodies an ALU to evaluate the proper router location. The NEWS (North, East, West, South) grid allows processors to exchange data between PEs which are physically connected. The direction to which data will be received, or transmitted, via the NEWS grid is determined globally (i.e. all the PEs will send, or receive, data from the same direction). Communication through the NEWS grid is faster than the router network if the destination for the data is a PE's four nearest neighbours.

Pipeline Image-Processing Engine (PIPE)

The PIPE system is a multistage pipeline architecture designed by the National Bureau of Standard (USA) for performing low-level image processing at TV-field rate. The system is built up in stages, as shown in Fig. 2.9, and excepting the input stage and the output stage, other processing stages are identical. The input stage contains two image buffers, used to store images captured from the input devices, whilst the output stage is employed to interface with the output devices. Between the input stage and the output stage are the Modular Processing Stages (MPS) which are responsible for the processing of the input image.

The most important features of the PIPE system are its multi-input and multi-output data streams, staged operation sequence and parallel neighbourhood operators. As shown in Fig. 2.9, the three data streams carrying data representing three 8-bit 256x256 pixel images are the forward path, backward path and the recursive path. The forward path is for passing data from one MPS to the next in the pipeline whilst the backward path passes data in the opposite direction. The recursive path conveys processed data back to a MPS. The advantage of such an arrangement is that it allows image data to participate in temporal, as well as spatial, neighbourhood operations.

Operations performed by each MPS can be divided into three stages. In the first, Boolean or arithmetic operations are applied to each of the input data stream. Operations are performed independently, and simultaneously, on each input. After the

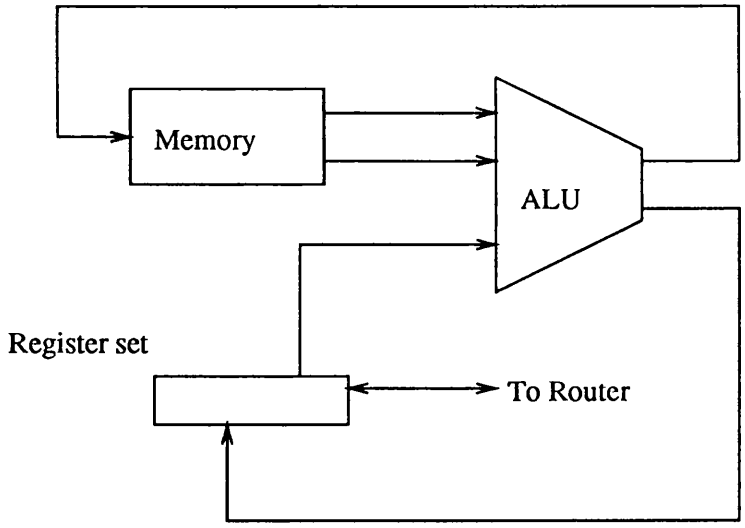


Figure 2.8 Block diagram of the Connection Machine PE

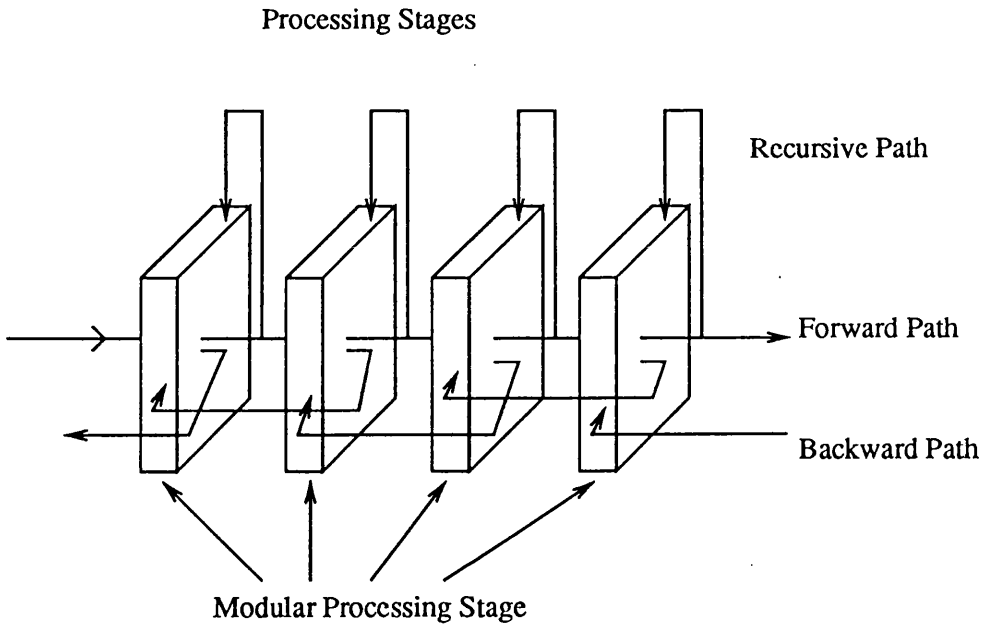


Figure 2.9 Block diagram of the PIPE system

operations, the three data streams are combined and stored in an image buffer. Neighbourhood operations, the second stage, follows. There are two neighbourhood operators which can carry out either arithmetic or Boolean operations in a 3x3 neighbourhood in parallel. Both operators receive identical input data but can perform different operations. Before the data leave the MPS, further transformation or arithmetic operations are possible. In the final stage, operations are applied to either outputs from the two neighbourhood operators by one of two programmable functions: a lookup table mapping function and an ALU performing arithmetic function. The data stream arising from either of these operations can be selected to become the forward path, backward path or the recursive path. As each MPS operates in parallel, real-time low-level processing is achieved.

PUMPS

The PUMPS system, proposed by Briggs et al of the Purdue University, is designed as an integrated image analysis and image database processing system. The structure of the system, as shown in Fig. 2.10, can be divided into an image processing and analysis unit, and the Database Machine. The sections are linked by data busses and shared memory.

The image processing and analysis unit embodies two kinds of processing elements namely the Task Processing Units (TPUs) and the Peripheral Processors and VLSI Units (PPVUs). Each TPU consists of a microprocessor and memory, and can perform three functions:

1. Dispatch and initiate PPVU tasks;
2. Execution of purely sequential tasks;
3. Participation in MIMD processes and running the operating system.

Communication between the TPUs is via the Task Processor Communications (TPC) Bus.

The relationship between the TPUs and PPVUs can be regarded as masters and slaves. The PPVUs are tailored for specific applications. Basic VLSI arithmetic modules for local matrix inversion and multiplications have been proposed. Other modules, such as the FFT processor, VLSI arrays or pipeline processors can also be incorporated. The TPUs gain access to the PPVUs through a switch network known as the Special Resource Arbitration Network. To avoid contention when several TPUs reference the same PPVU, TPUs are allocated a PPVU on a priority basis. The TPUs also share the Shared Memory (SM) and the Shared Cache. The Shared Memory is

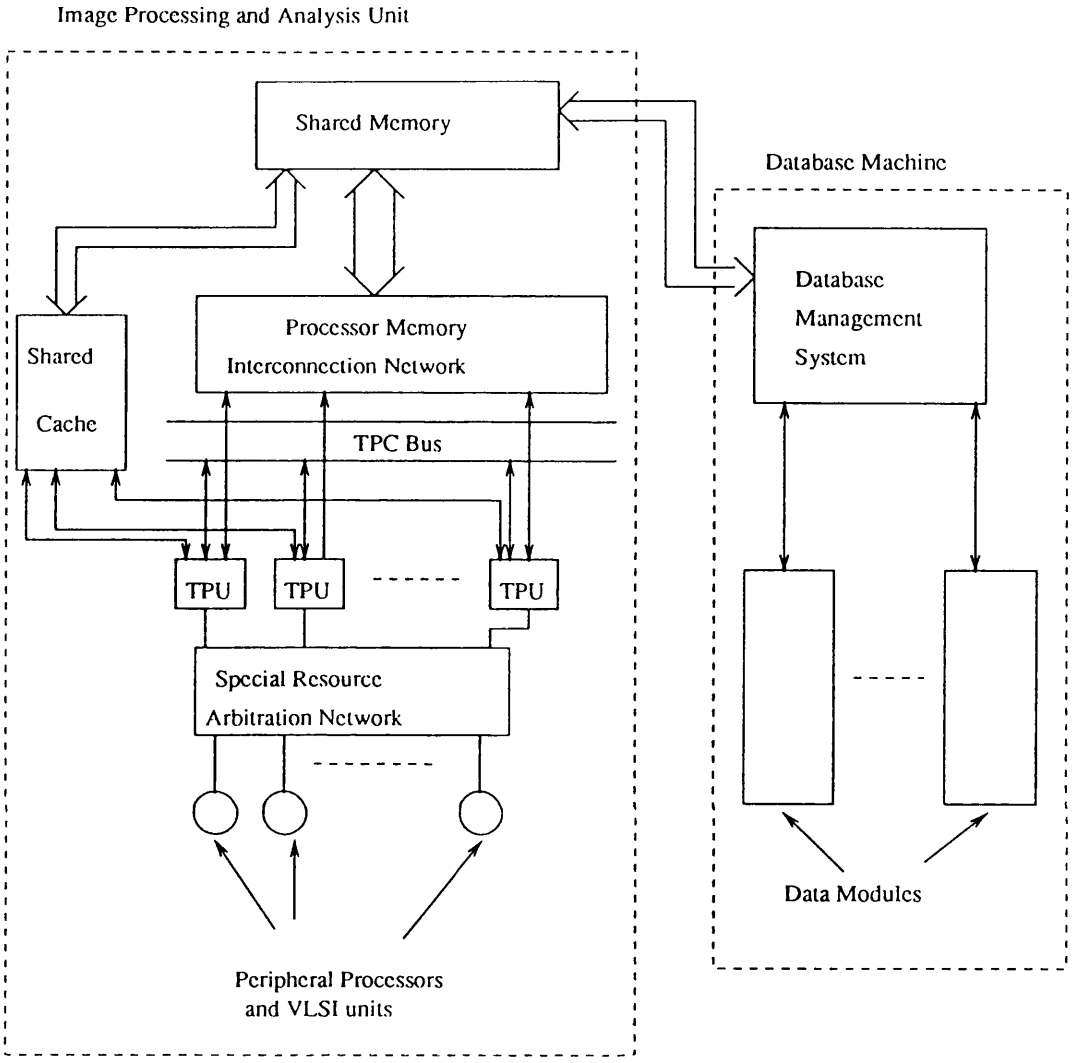


Figure 2.10 Architecture of the PUMPS system

used as a data transfer and storage medium. Blocks of data can be transferred from any TPU to the SM through the Processor-Memory Interconnection Network. The Shared Cache is employed as a temporary storage for data which is operated on by the TPUs because of its high speed. The Shared Memory is also used to exchange data between the Database Machine and the image processing and analysis unit.

The Database Machine is designed to support high speed image retrieval from the storage device by enabling a user to specify relational information of an image. To achieve such a requirement, the Database Machine supports both high-level database functions and low-level image processing operations. The major components in the Database Machine are the relational database system for images (REDI) and the relational database machine (DIALOG). The REDI supports a relational query language which enables a user to select an image from the database by specifying the spatial relationships between objects in an image. The DIALOG performs the processing required by the enquiry and embodies identical data modules operating in parallel. Each data module consists of an image storage device, a disk, and special hardware to perform the particular database, or image, functions. This currently includes join, selection and histogramming.

2.4 Discussion

In the above sections, different multi-layer and multi-processor systems have been studied. Each system incorporates one or more unique features, each of which improves the system performance in specific circumstance. A brief summary is given here.

Pyramid systems can be divided into homogeneous and heterogeneous systems. A homogeneous pyramid employs one type of processing element at all layers. The systems studied have the following common features:

1. Size of base level equals to the input image so that each pixel can be processed by one PE.
2. A very simple element is used, usually single bit, meaning that several PEs can be housed in a chip (this, in return, reducing the number of chips employed in the overall system).

Two controlling mechanisms are used by the homogeneous systems, SIMD and multi-SIMD. Basically, PEs in the same layer perform the same operation but layers can carry out different processing tasks. All, except the HCL pyramid, provide a feedback signal reflecting the status of PEs in a layer, this improves the efficiency of a

system.

Heterogeneous systems employ different processing elements at different layers. Generally speaking, a layer in a heterogeneous pyramid performs a particular level of processing tasks and there are only three layers in a system (excluding the host). The two systems presented also include the following common features:

1. Base layer has a size equal to that of the input image and a simple PE, with 1-bit ALU, is employed
2. Different processing elements are employed at different layers, the complexity increasing with the level of the pyramid.
3. Shared memory (dual-ported) is used as a communication medium between layers.
4. Top-down control, meaning that elements in one layer can instruct elements in the layer below, is allowed.

The controlling mechanism used by the heterogeneous systems is more complex. Elements in the base layer can be operated in either SIMD, or multi-SIMD, mode. In the latter case, elements in the middle layer act as the controller for the base layer. The upper layers are under MIMD control.

The survey demonstrated that pyramid architecture suffers from certain shortcomings with the implication that it is not suitable for the envisioned system. As discussed in Chapter 1, the envisioned system will embody three layers, with each layer performing different kinds of computation. Homogeneous pyramids usually consist of $(\log_2 N + 1)$ layers, when there are N^2 PEs in the base layer, and are most suitable for performing multi-resolution processing [30]. Homogeneous pyramids, under SIMD control, cannot provide functional parallelism.

In comparison, the heterogeneous approach seems more suited for the envisaged system, but several design problems arise. In order to achieve efficient image to image transformations (low-level processing), a one PE per pixel mesh-connected array serves as the base layer. The convergence ratio between layers is high (64 and 256 for the two systems), so a large quantity of communication and control channels is required between elements in different layers. This, in turn, increases the complexity of a system. Even though a high convergence ratio is applied, the number of PEs available at the middle and top layers is still high. As powerful computing units (16-bit and 32-bit devices) are employed by these layers, the cost of a heterogeneous pyramid is likely to be high. Another problem relates to the programming of the system. Different computing units are incorporated in different layers and the basic

programming requirements for them will be different. It is, therefore, more difficult to develop the system software.

Images are the first data type to be processed in any vision system, making the mesh-connected array an obvious choice as the basic architectural unit of a multi-layer system. However, in other levels of processing, different data types, for example symbols and feature lists, are involved and these may not map well onto the 2-D array. This suggests other architecture(s) should be provided in the middle and top layers, or alternatively that an architecture which can be adopted for different data types should be used as the basic structure. Linear array is one such alternative and will be examined in the next chapter but before that, the properties of other multi-processor systems are studied.

Other multi-processor systems all have their own special features which may be adopted by the envisioned system. First, the Connection Machine which, as its name implies, emphasises the importance of inter-element communication by providing a 12 dimensional hypercube router network. The system can be visualised as a two layer machine, with its PEs in the base layer and the router network as the second layer. Data from any PE can be transferred through the router network to any other PE within the system.

The special design feature in the PIPE system is the multi-input and multi-output data streams. The three input data streams, forward, backward and recursive provide temporal, as well as spatial, information. It is, however, possible to interpret the multiple data streams as another form of inter-element connectivity. Another special property is the parallelism permitted within each processing stage. Three different operations can be applied to the inputs in parallel and two different neighbourhood operations can be performed at the same instant.

The PUMPS system demonstrates a rather direct approach to the construction of a highly parallel system. In the proposed system architecture, different computing units such as pipeline processors and a FFT processor, can be incorporated in the system. There is also a special database machine for image. Communication between elements is via switching networks or shared memory. The drawbacks of the system will be its cost and complexity: the cost is proportional to the number of components included and the different controlling mechanisms required by the various components impose a penalty on development of system software.

From the above discussion, communication (or connection) between elements in a multi-processor system is an important design factor, this is reflected from the special arrangement provided in the Connection Machine, the PIPE system and various

pyramid systems but the different communication networks described are designed in a somewhat generalised manner. Of the various systems presented, many are designed for a particular class of problem. The WPM, for example, is intended for image understanding and computer graphics, whilst the PUMPS system was designed for image database processing. These systems provide special processing units in order to cater for the different computing requirements imposed by a problem. Besides computing requirements, communication networks between elements should also be designed according to the structure of a problem. This will be a major area of investigation in the following chapters.

Chapter 3
Linear Arrays

3.1 Introduction

The objective of this thesis is to develop a multi-layer system consisting of three layers performing low, intermediate and high level processing independently. The envisioned system can be constructed by adopting the heterogeneous pyramid architecture but this does suffer from certain shortcomings including cost, complexity and programming methodology. Additionally, the data types evolved in higher processing levels, unlike an image, do not have an intrinsic structure; this suggests that elements in those levels do not need to be arranged as 2-D arrays and linear arrays serve as one alternative. As stated by Basille et al [15], and reflected by the many linear image processing systems being built or proposed (see Section 3.3), linear array can be applied to both image based and non-image based processing. It is, therefore, possible to employ the linear array as the basic architectural unit for the envisaged system.

The linear structure is chosen because it has (as described in [31]) the following advantages:

1. Flexibility
   The reduction in the number of PEs embodied in a linear array implies that each PE can have any desired degree of autonomy (this is treated in Section 3.2). This flexibility widens the operational scope of an array.
2. Ease of construction
   In a linear array each element is (usually) connected to its left and right neighbours in the string. This facilitates extension to a larger number of processors.

The physical implementation of a linear array is easier than that of a mesh-connected array due to the simpler connectivity employed. Each element is connected to its left and right neighbours and this connection network is quite easy to retain physically, whilst in a mesh-connected array, nearest-neighbours connectivity is difficult to retain physically and neighbourhood connections can often only be achieved via external cables. In the context of the construction of a multi-layer system, a multi-linear array system is a 2-D structure, which is difficult to build because of the inter-layer connection network. In comparison with the 3-D structure of a pyramid system, however, a multi-linear array system is much easier to construct. In many cases, a linear array with N PEs is employed to process N^2 pixel images, implying a much smaller number of PEs available in the base layer than in the pyramid systems. This leads to a reduction in the number of communication channels required between the elements in the

base and intermediate layers, implying that a smaller number of PEs can be used in the intermediate layer and a similar effect applied to the top layer as well. As a result, both the cost and complexity of the multi-layer system will be less than that of the heterogeneous pyramid approach.

Linear arrays have other advantages including expandability and fault tolerance, the latter being an important factor that should be included in any computer system. In a multi-element parallel computing system it is possible to embody spare processing elements in the system in order to maintain the functional size of an array when elements fail. Elements in an array are connected to their neighbours and it is important to retain such connectivity, especially in the low-level layer, after a faulty element is switched off. In a 2-D array, the re-establishment of the near neighbours connection requires a very complex switching network, as described in [32]. One strategy by which the switching network can be avoided involves switching off the whole column or row that contains the faulty element. In this case, only the communication channel in one direction needs to be re-connected, as shown in Fig. 3.1. This implies that extra columns/rows of processing elements are needed, but, for a linear array, the communication between elements is along a single direction and the switching off of a faulty element requires the re-establishment of one communication channel only, as shown in Fig. 3.2. Individual faulty elements can be bypassed and replaced by spare processing elements connected to the end of the array.

A linear array can be expanded by the addition of extra elements at the end of the array due to its one dimensional property and simple neighbourhood connectivity. This property is valuable for the second and third layers of the system because the amount of data involved in those layers is variable (see Chapter 6). Expanding an array to improve parallelism may be necessary and this will be discussed in Chapter 7. The rest of this section will demonstrate how a linear array can be applied to different levels of processing operations.

In low-level processing, the 2-D structure of an image can easily map into a linear structure. For example, a linear array with N elements can process images with N^2 pixels (it is assumed that each element has sufficient memory to store the pixel data lying in the same image column). A row of pixels will be processed at each operation cycle, so it will take N cycles to complete a point-wise function. In a linear array, neighbourhood operations can also be carried out easily. The numbering convention for a pixel's eight nearest neighbours is shown in Fig. 3.3. This orientation applies to all nearest neighbour definitions throughout this thesis. In a linear array, the pixels within a 3x3 window come from the PE itself as well as its left and right-hand

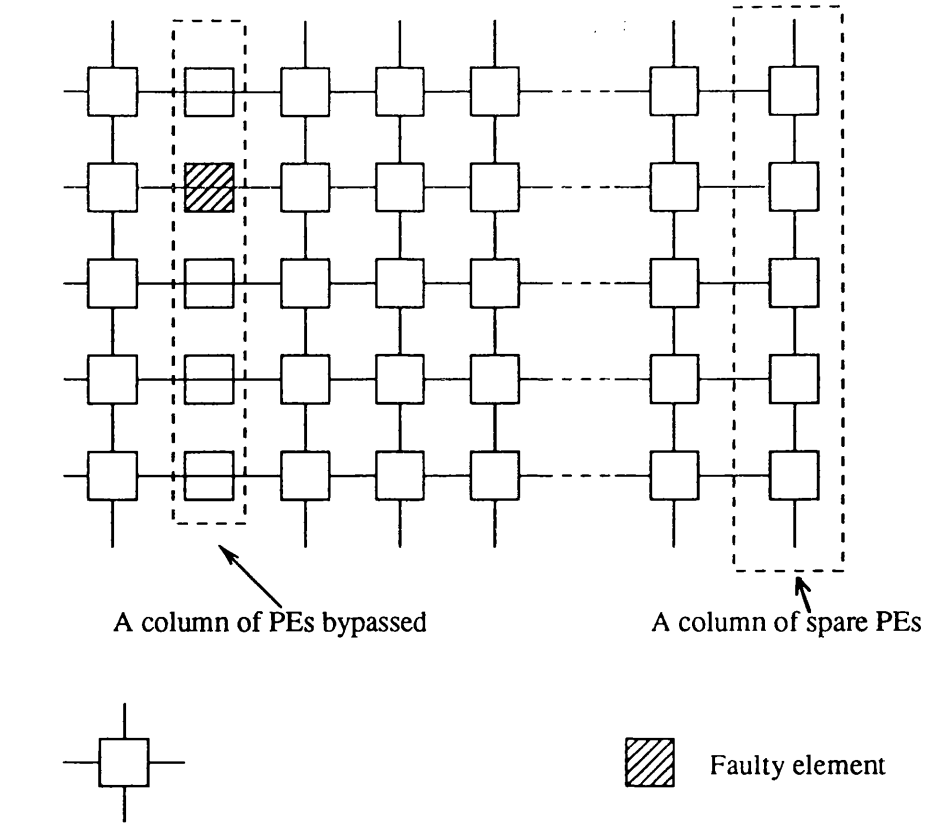


Figure 3.1 Bypassing a column of PEs in order to achieve fault tolerance in a mesh-connected array

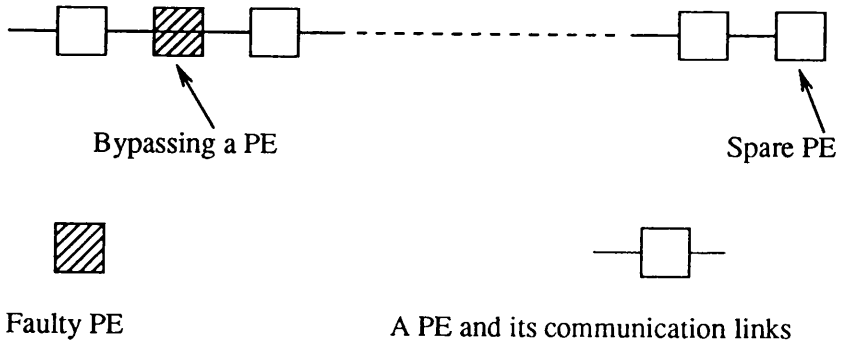


Figure 3.2 Bypassing a PE to achieve fault tolerance in a linear array

neighbours. During a neighbourhood operation, two consecutive windows have six pixels in common, as depicted in Fig. 3.4, and consequently only three new pixel values are needed in order to evaluate the result for the second window. Two of the three pixels are located in the neighbouring PEs and can be extracted easily with two shift operations. The third pixel comes from the PE's local memory.

As described in Chapter 1, images are processed by the low-level array, whilst the intermediate layer operates on symbolic data. It is, therefore, necessary to transform information conveyed by an image, after low-level processing, into a different format (symbolic) before intermediate-level processing proceeds. A linear array can also easily re-arrange information carried by an image to a symbolic data matrix because each PE can access any pixels residing in its local memory in a read operation. This simplifies the process of gathering information from different locations in an image in comparison with the many shift operations required to transfer data between PEs in the same column in a mesh-connected array. A PE can then extract the useful information, which will be stored in the local memory. In such an arrangement, parts of a PE's local memory carry image-based information (pixels), whilst symbolic data is available from other parts. Only the symbolic data is transferred to the intermediate layer and, because, in most cases, the quantity of symbolic data is less than the number of pixels in an image, this in turn reduces the rate of communication between the two layers.

It is difficult to indicate the effectiveness of a linear array for intermediate-level and high-level processing, due to the absence of a consistent data structure involved at these levels. That the data structures which evolve depend on the particular vision problem will be demonstrated in Chapter 6, but in general a variable appearing in intermediate-level or high-level processing can be expressed as a string of data [31] rather than as a single pixel value. Each element in a linear array can either process data belonging to the same string or each element can operate on a section of one string so that parallelism can be achieved.

It has already been suggested that the linear array can be adopted for image processing at different levels. The linear array is easier to construct than a mesh-connected array and a multi-layer system constructed from linear arrays is correspondingly simpler than a 3-D structure (pyramid structure). Linear arrays are also easier to expand and to construct with fault tolerance. It is thus reasonable to conclude that they can be employed as the basic structure of a multi-layer system.

1	2	3
8	X	4
7	6	5

Code	Direction Relative to Pixel X
1	North West
2	North
3	North East
4	East
5	South East
6	South
7	South West
8	West

Figure 3.3 Numbering convention of a pixel's 8 nearest neigbours

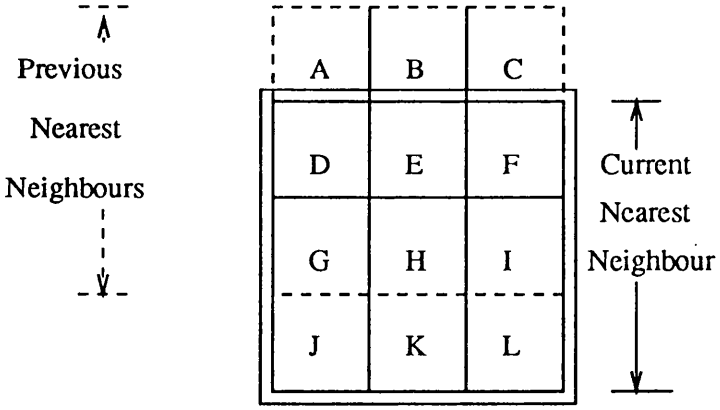


Figure 3.4 Two consecutive 3x3 neigbourhood operators

3.2 Local Autonomy

In Section 3.1, the linear array was proposed as the basic structure for a multi-layer architecture and its suitability is supported by the fact that a linear array can be applied to processing at all levels and is relatively easy to build. By employing a smaller number of processing elements in the system its performance may obviously be affected, especially in the low-level operations: it is therefore desirable that the performance be improved through other methods. The introduction of local autonomy to elements in an array is one such solution.

As discussed in [33], there are six levels of autonomy available to each processor in an array comprising activity, data, function, operation, sequencing, and partitioning.

1. Local activity control is the ability to determine locally whether or not each processor is active.
2. Local data control applies to processors which can locally select the source and destination of the data used by a globally-determined process. Source and destination can be located in the local memory or a neighbouring element.
3. Local function control allows a processor to select which function is to be performed under pre-determined conditions.
4. Local algorithm control enables a processor to execute an algorithm resident in the local programme store. The sequencing of these stores is under global control, thereby ensuring synchronous operation of the array.
5. Local sequencing control is an enhancement of local algorithm control. Each processor can sequence its own programme memory which stores the code of the algorithms. Under this mode of autonomy, synchronisation between elements is no longer automatic, but must be ensured by appropriate handshaking operations.
6. Local partition control is the highest level of local autonomy. At this level, a processor transfers programme segments to other processors in order to share its work load with other elements.

In order to incorporate the different levels of autonomy in a system, modifications to the structure of the processors are required. Estimates for the cost, performance, and processor complexity imposed by different degrees of autonomy are found in [33].

Different systems embodying various levels of local control are being built or proposed and in the following section some of those systems are described. A survey of existing or proposed linear array systems, with the aim of attempting to establish a better understanding of their structures, is made. The survey considers the complexity of the processing element, the degree of local control, controlling mechanism and

programming methodology.

3.3 A Survey of Linear Arrays

Existing linear arrays are of differing complexity and it is difficult to cover every detail of the arrays included. The survey therefore tries to highlight the important features embodied.

AIS 5000

The AIS 5000 [34] is a SIMD linear array, developed by Applied Intelligent Systems Inc. The AIS 5000 is a standalone system comprising three units: a Motorola MC68000-based host, a high speed I/O unit, and the parallel array unit. The host provides storage for the user programmes and essential peripherals for the system. The high speed I/O unit provides an interface between the host and the array processor. Moreover, video monitor output and camera input are also connected to it. The other function of the high speed I/O unit is to convert the data from byte serial format from I/O devices to bit parallel for the local memory, and vice-versa, and this is performed by the corner turner. There are three corner turners, two for outputting data from the memory and one for inputting.

The parallel array unit comprises both the processing elements and their controller. The linear array can include up to 1024 PEs, each of which embodies five major components: a Boolean operations unit; a neighbourhood operation unit; an arithmetic operations unit; an I/O unit and 32 Kbits of local memory, as shown in Fig. 3.5. Both the Boolean and neighbourhood operations units are programmable and through lookup tables. Functions performed by those units are dependent on the values of the relevant lookup tables supplied by the user. The values of a lookup table are specified by a 16-bit pattern. The Boolean unit allows up to 4 inputs, whilst the neighbourhood operations unit will receive input from its four nearest neighbours. Data arriving from the north, south, and centre is automatically shifted when a new row of image is read. The arithmetic operation is performed by a single bit ALU producing both a result bit and a carry flag, and this enables multi-bit arithmetic.

There are three I/O channels, operating at 7MHz, which facilitate the input, or output, of data from, or to, the parallel memory. The I/O channels operate independently of and asynchronous with the processing element. The I/O channels access the local memory via a non-maskable interrupt and the direction of data transfer in each channel is independent. The PEs, including the I/O channels, are implemented by a custom-designed integrated circuit in which 8 PEs are accommodated.

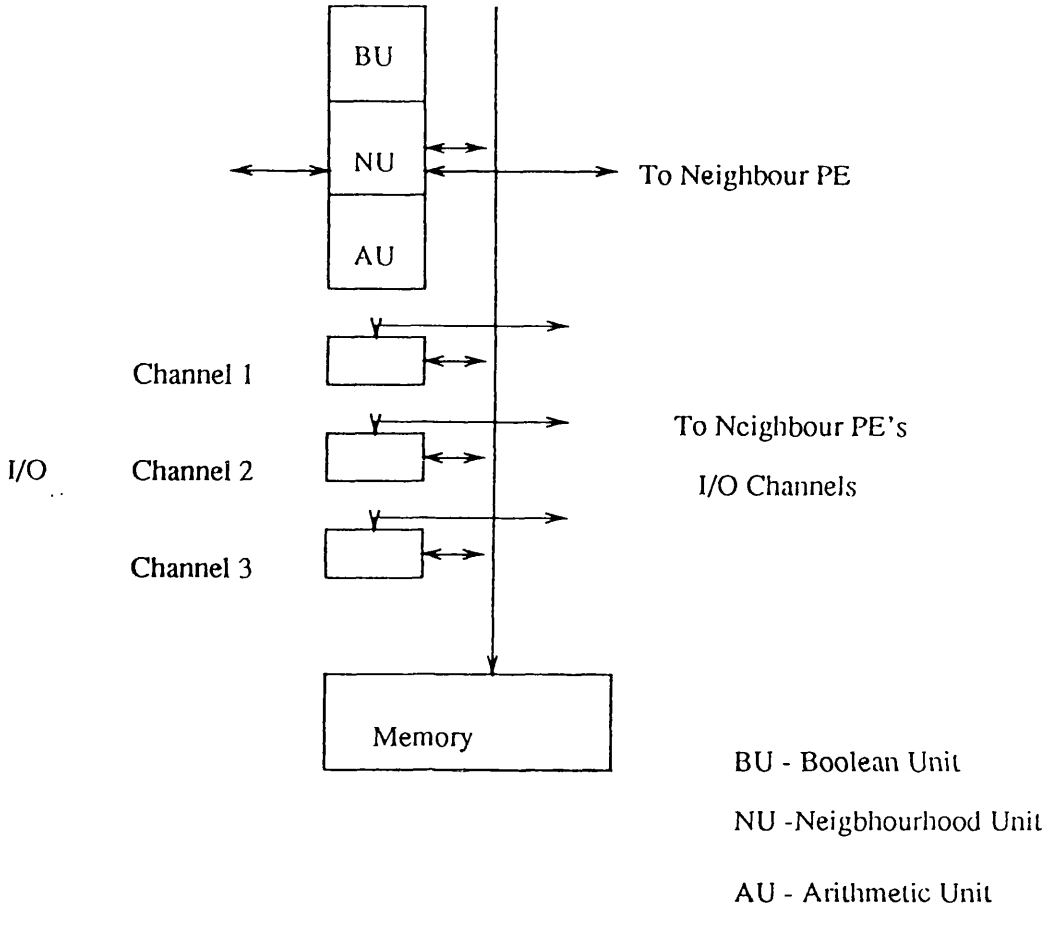


Figure 3.5 Block diagram of an AIS 5000 PE

The control signals are issued by the array controller, which also broadcasts the addresses for the local memory. The system can be programmed in C language and various operations of the array processor are implemented as a function library.

ASP

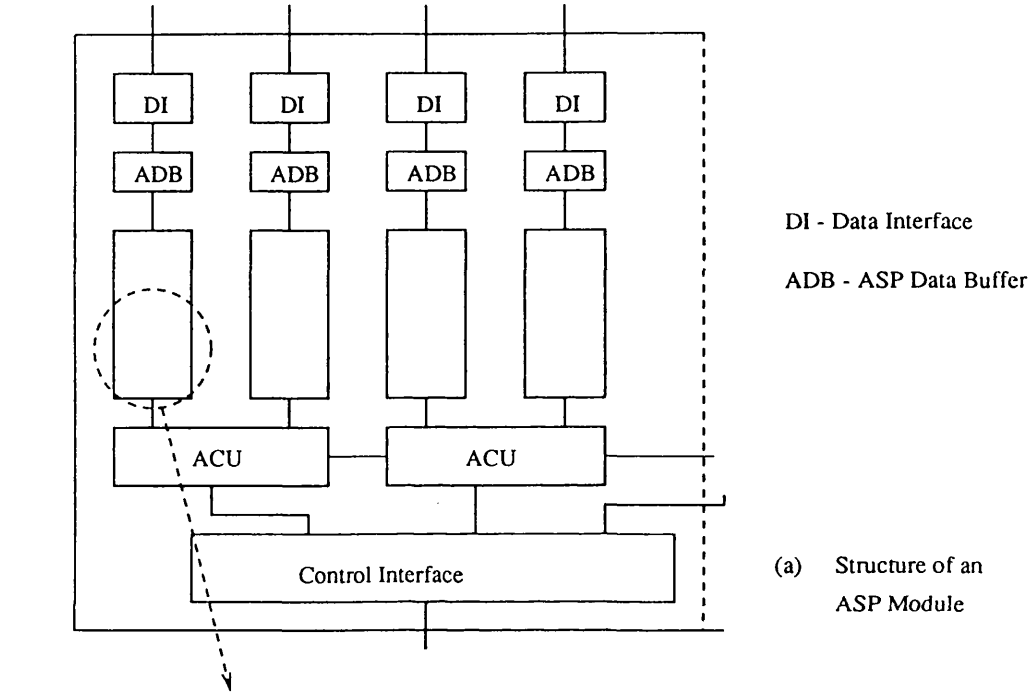
The Associative String Processor (ASP) [35] is a very complex system devised for general purpose processing. The ASP architecture can be classified into three levels of complexity, which are the ASP modules, the ASP substring and the Associative Processing Element (APE). An ASP system will comprise a number of ASP modules, each of which embodies a group of ASP substrings, as shown in Fig. 3.6(a). Each ASP substring incorporates a string of APEs, a ASP controller (ACU), an inter-APE Communication Network, and a Data Buffer. The APE (see Fig. 3.6(b)) is a simple 1-bit processor which supports five basic operations: match, add, read, write, and shift. APEs within a ASP substring share a bit-parallel Data bus, Activity bus, a Control bus, and a single feedback line (Match Reply). The APEs are under SIMD control and instructions are broadcast from the ASP controller via the control bus. The most special feature of the ASP system is the Content-matching mechanism. An APE is selected, for subsequent parallel processing, by comparing the content of their data and activity registers with that of the corresponding Data and Activity busses and will only be active only if there is a match.

Communication between APEs can be achieved either through the inter-APE Communication Network or the shared Data Bus. In both cases, only activity signals are transferred between the APEs thus taking advantage of the content-matching property of the system. The inter-APE communication Network is reconfigurable and this enables the emulation of other common network topologies.

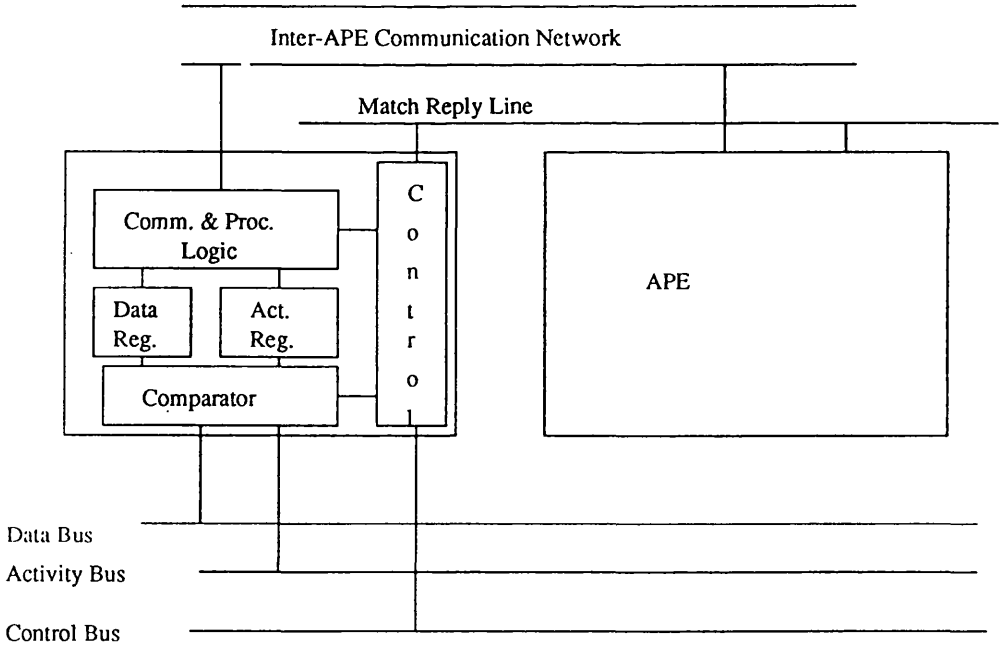
The system can be programmed in high-level block-structured languages such as: Pascal, C, etc. These programmes will invoke the ASP subroutines. The implementation of the ASP will exploit wafer scale integration technology and a 65536 APE ASP module is proposed.

CLIP7A

The CLIP7A [36] system is developed at University College, London and comprises 256 processing elements each of which consists of two CLIP7 chips, 64K RAM and auxiliary logic, as shown in Fig. 3.7.



(a) Structure of an ASP Module



(b) Structure of an ASP substring

Figure 3.6 Block diagram of the ASP system

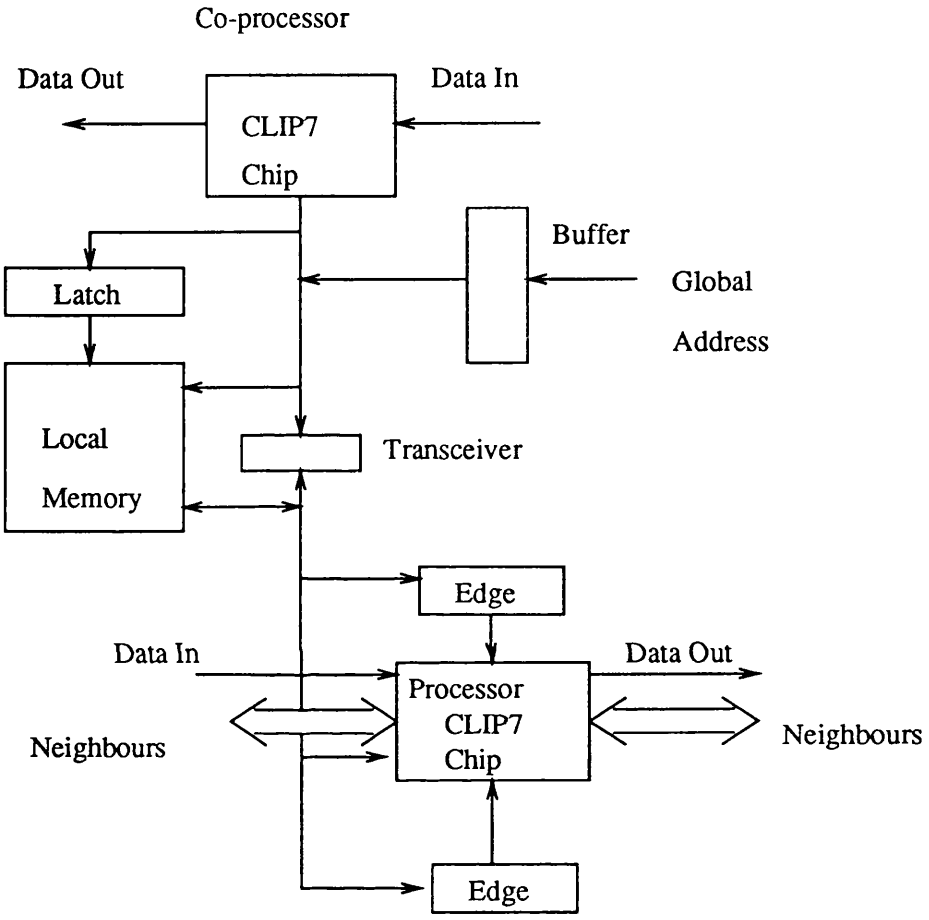


Figure 3.7 Block diagram of a CLIP7A PE

The CLIP7 chip is a custom-designed circuit implemented in 5μm CMOS technology with a single processor in each chip, details of which can be found in Chapter 4. The chip embodies a 16-bit ALU; a set of 4 scratchpad registers; a shift register and three types of special purpose registers. The special purpose registers consist of a D-register, Neighbourhood registers and a condition register. The D-register is assigned for the purpose of data transfer between elements. The Neighbourhood registers provide storage for data from within a 3x3 window. The value stored in the condition register can affect the operation of the ALU and can enable/disable the loading of various registers.

The CLIP7A system can be classified as a semi-SIMD machine because each processing element is capable of generating local RAM addresses; selecting data from the Neighbourhood registers and conditional operations. The two CLIP7 chips are named processor and co-processor, as shown in Fig. 3.7. The processor is responsible for numerical operations, while the co-processor is used to generate RAM addresses. The addresses for the local RAM can be generated in three different modes: global broadcast, half global and half local, and fully local.

The communication between PEs is via two uni-direction 8-bit channels called D-chains, formed by connecting the D-registers of the processors together. All the co-processors' D-registers form one D-chain, whilst the other is formed by the processors' D-registers. Both pass data in a single direction only.

The two edge registers are designed to hold data from the rows both on top of, and below, the current processing pixel. Each set of three consecutive PEs, therefore, contains data lying within the 3x3 window. The edge registers are connected to the Neighbourhood registers so that data within a 3x3 window can be stored in the Neighbourhood registers. Each PE can locally determine the direction from which data will be retrieved from the Neighbourhood registers.

The lowest level programming language is by micro-code and various routines written in micro-code can be invoked through a higher level language, called C7VM (CLIP7 virtual machine) [37] which is an extended version of the C++ language [38].

LAP

The Linear Array Processor (LAP) [39] was proposed by the National Physical Laboratory as a low cost image processor for robotics and automated inspection. The number of processing elements employed by the system equals the width of the input images and each has a fast single-bit Boolean processor; 16 single-bit working registers; a 256-bit RAM and a 8-bit I/O register, as shown in Fig. 3.8. A complete system

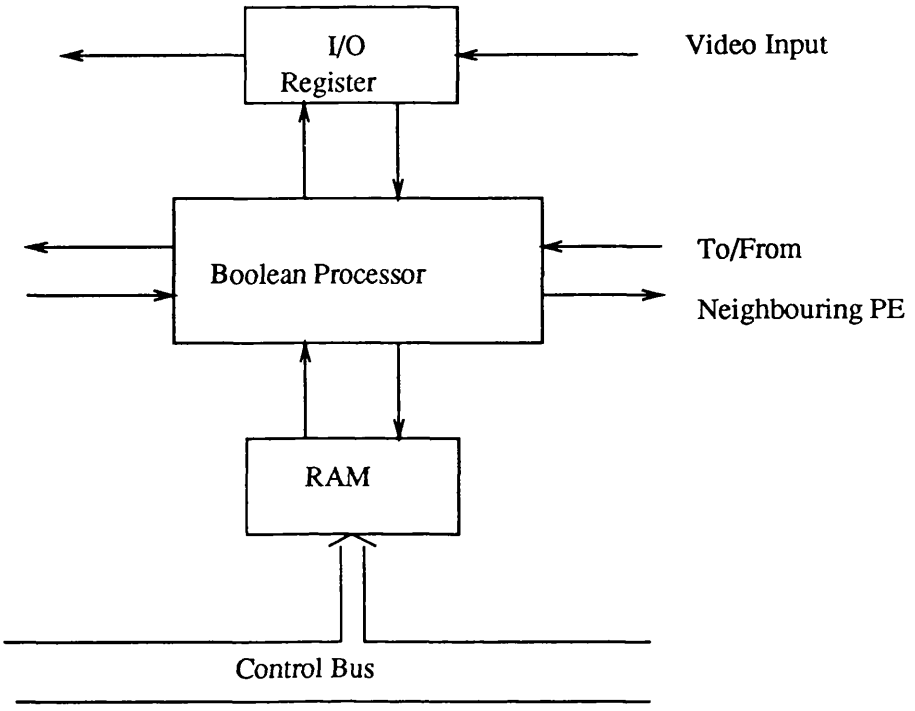


Figure 3.8 Block diagram of a LAP PE

may consist of up to 256 PEs. The control of the system is SIMD and the operations of the LAP are defined by 24-bit microcodes. A prototype system was built from bit slice devices but no further development has been reported.

PICAP3

The PICAP3 [40] is a product of Linköping University. It is a 64-element linear array designed to performed three-dimensional image processing operations. The structure of the system can be divided into three parts: a host computer, a controlling unit and the linear array.

The linear array consists of 64 Processor Memory Elements (PMEs), the Structure of which is shown in Fig. 3.9. The major components embodied in the PME include a 16-bit ALU; a 16x16 multiplier; 4 Mbytes of local memory and I/O buffers. The 16-bit ALU employed by the system is the American Micro Devices' AM29116 microprocessor and the multiplier is the Analog Devices' ADSP-1110, which generates a 32-bit product for a 16x16 bit multiplication. The memory is organised as 1 Mwords by 32-bit format and is accessible by external devices via the global I/O bus. Each PME is connected to its two nearest neighbours and communication between them is carried out through two unidirectional channels each of which is 16 bits wide and formed by connecting the neighbourhood register in each PME in a chain. The control of the array is basically SIMD but local data-dependent activities are permissible. The activity bit can deactivate the operation of a PME whilst the ALU is capable of modifying the globally broadcast memory addresses.

The controlling unit embodies two major components: the system controller and the supervisor processor. The system controller incorporates a microcode sequencer, a 2Kx160-bit microcode programme memory and an address and constant generating unit. The controller broadcasts microcodes, memory addresses and constants to the PME array. The PME array is controlled by 46 bits of the microcode and the remaining bits are used to operate the controller itself. The supervisor processor is a MC68020 microprocessor based unit, which exercises the highest level of control in the system. The unit monitors the operation of the control unit and hence the entire parallel processor. It also provides the communication channel with the outside world over the Ethernet and to a local disk, used for programme and data storage. The host provides the user interface and a software development facility. The system is programmed in microcode.

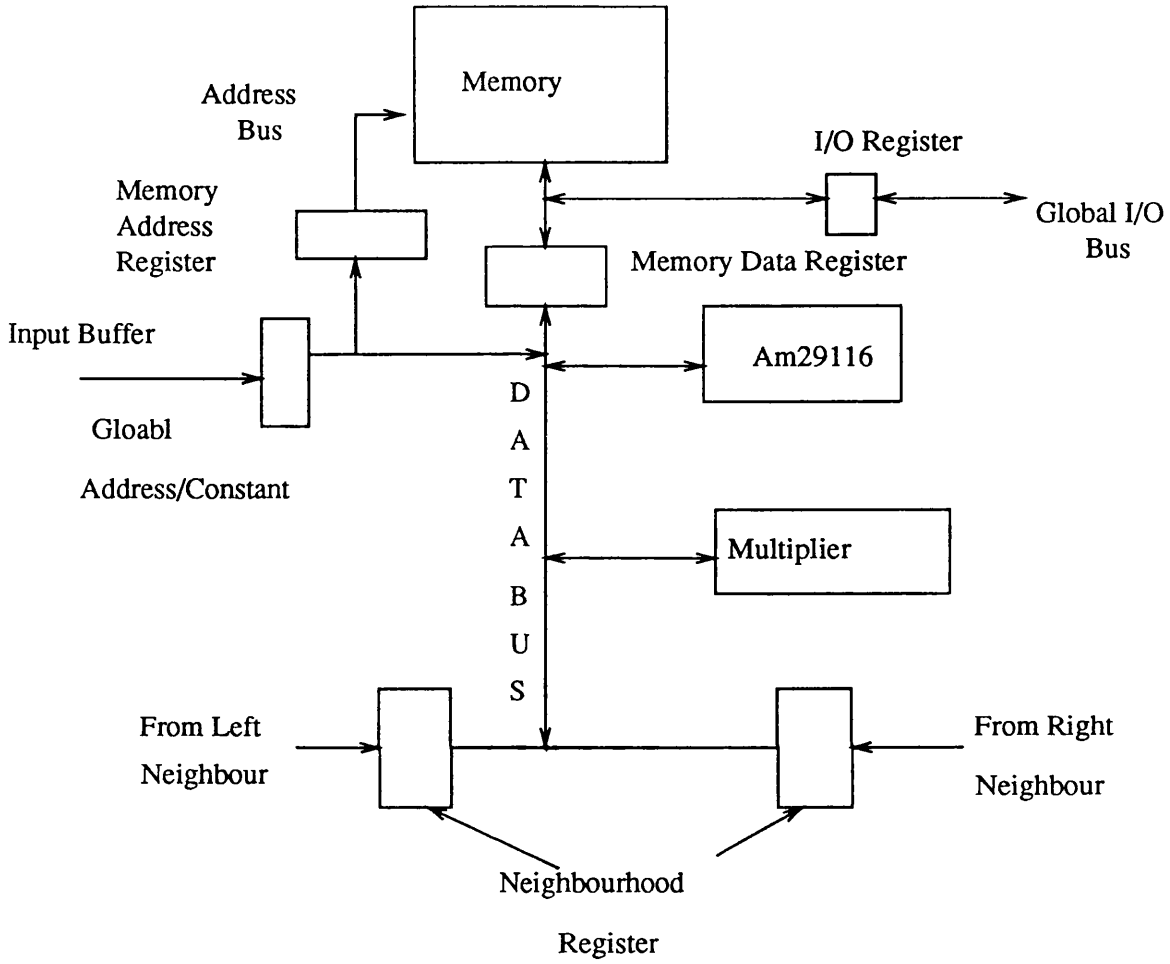


Figure 3.9 Block diagram of a PICAP3 PME

PIP

The Parallel Image Processing (PIP) [41] system is developed at the National University of Singapore and is an 8-processing element SIMD machine. Each processing element, depicted in Fig. 3.10, comprises three major components: a Texas Instruments' TMS32010 processor; 8 Kbytes of image buffer and 8 Kbytes of result buffer. The TMS32010 is a 32-bit microprocessor incorporating a 16x16 bit parallel multiplier, 144 words of on chip data RAM and 1.5 Kwords of on-chip programme ROM. The system is under SIMD control and the controller is a MC68000 based single-board computer. The operations of the PEs are performed according to the programmes stored in the programme memory which is a 4 Kword static RAM. There is only one programme memory and the data stored in it is broadcast to all the PEs. The programme memory is loaded by the controller during system initialisation and it is PE1 that issues addresses for the programme memory. PE1 is also assigned to generate addresses, broadcast to all the PEs, for the two buffers. Once initialised, the address for the buffers is automatically incremented by a special hardware device.

The system is devised to process 256x256 pixel images and each element processes a strip of an image (32 lines of 256 bytes). A PE can read data stored in the buffers of its adjacent neighbours through a data multiplexer but can only write to its own buffer. The result buffer is used to store the results after image operations and can be accessed by the frame display unit for outputting the processed result.

SLAP

The Scan Line Array Processor (SLAP) [42] was proposed by reseachers in the Carnegie Mellon University. The system is designed to achieve high speed and high bandwidth image input and output; a reduction in the size of the overall system and support of both iconic and higher-level image processing. The size of the system is not specified but a one PE per image column ratio is proposed.

The structure of a PE, as depicted in Figure 3.11, includes a 20-bit ALU; a register file; a rotate-shift unit (RSU); a neighbour-neighbour register (NNR); a video-shift register (VSR) and a response-line register (RLR). The components are linked by two 20-bit busses, designated A and B. The A bus is used for reading and the B bus for writing. The register unit contains 32 dual-ported 20-bit registers and one port is connected to the A bus whilst the other is accessed through the B bus. Such an arrangement permits the simultaneous writing to, or reading from, the registers. The ALU can perform Boolean operations, addition, subtraction, rotation and shift to its two inputs. Support is also provided for efficient multi-step integer multiplication and

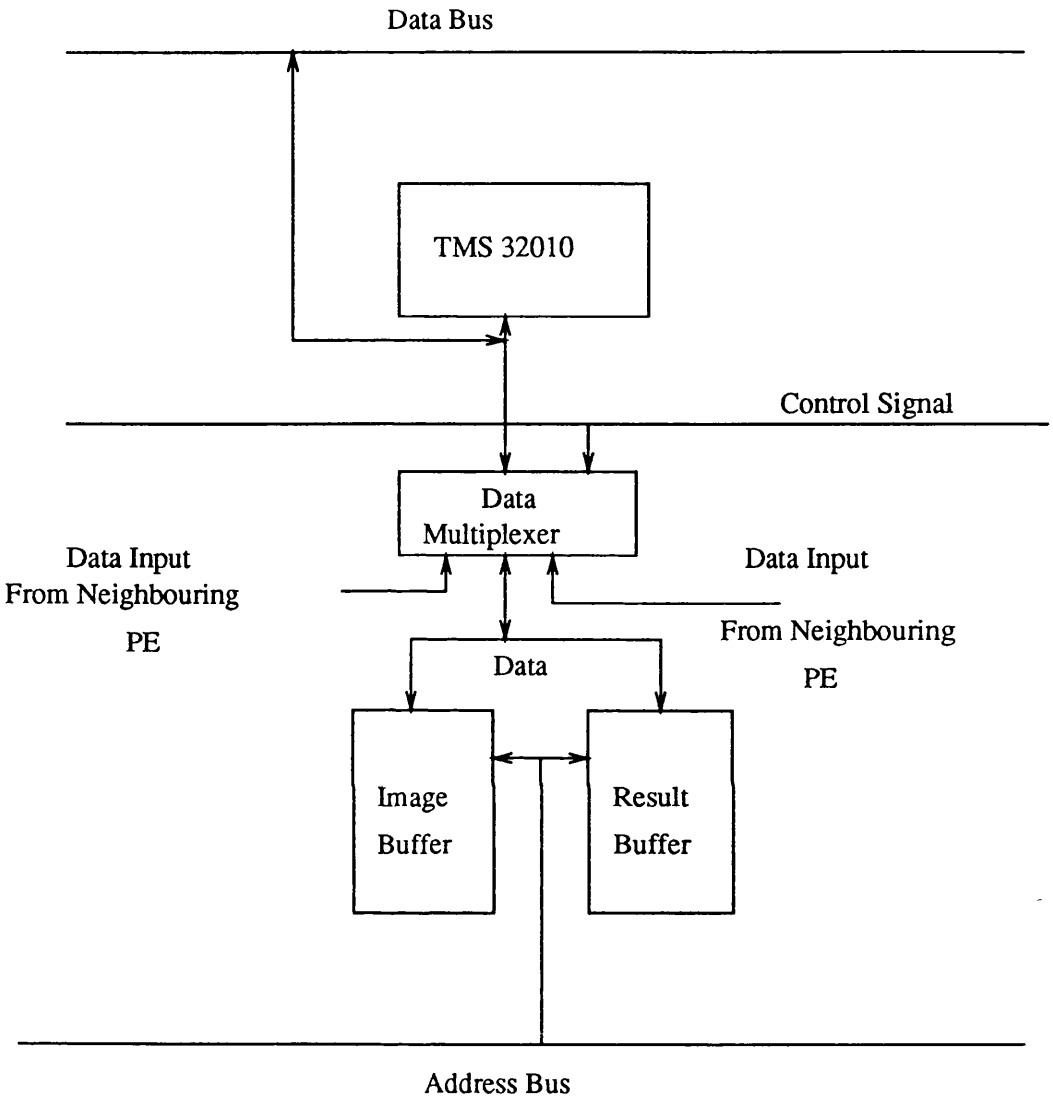


Figure 3.10 Block diagram of a PIP PE

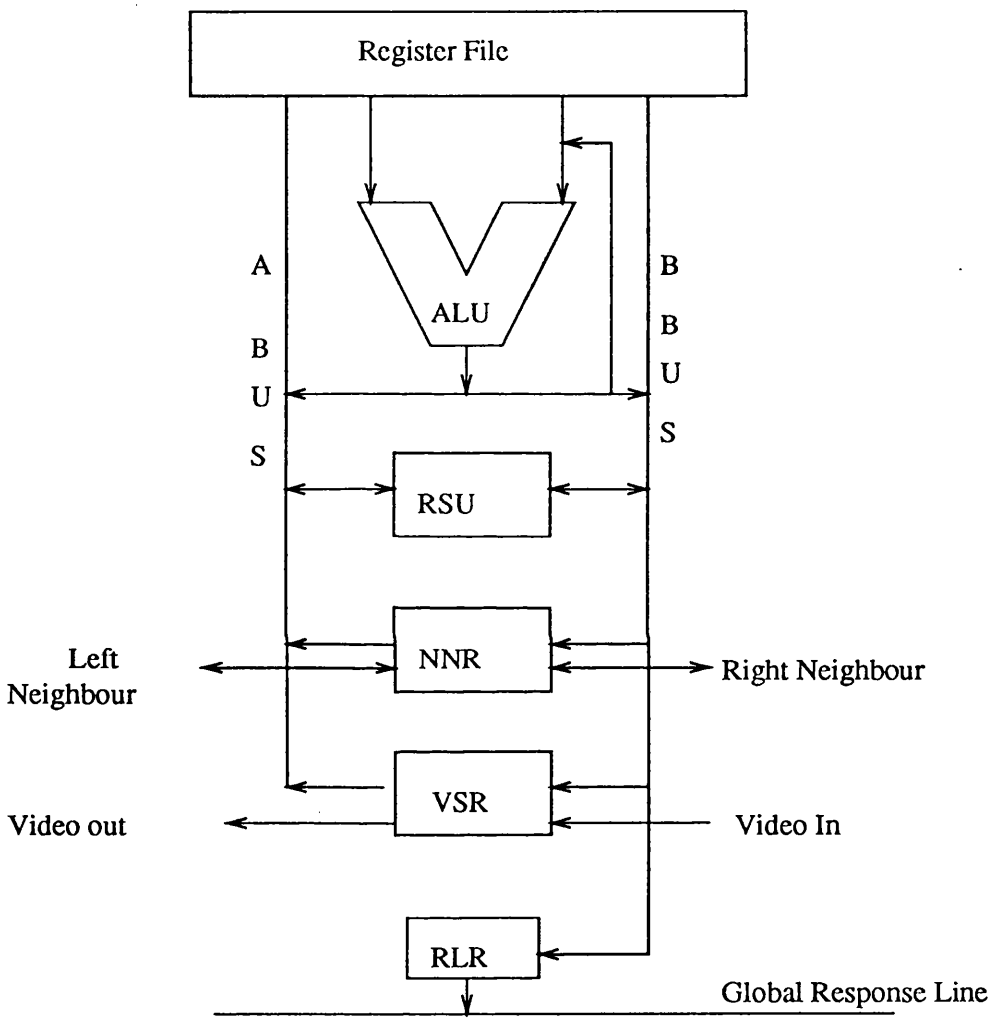


Figure 3.11 Block diagram of a SLAP PE

division. The RSU performs logical and arithmetic shift and rotate operations, whilst the NNR handles communication between adjacent PEs. The NNR in every PE is connected in series and it can be shifted in either direction, so that data can be transferred from one PE to another. The VSR is under global control and operates at a multiple of the instruction clock rate in order to produce a real-time image throughput. The RLR is a single bit register which writes to the wired-OR global bus, providing a feedback signal. The PEs will be implemented as a custom-designed circuit with 4 PEs residing in a single chip.

The control of the system is a combination of both vertical and horizontal microprogramming. An instruction comprises of a vertical field and horizontal fields. The vertical field is decoded on-chip, guiding data path activity over three successive clock cycles. The horizontal fields each dictates an action concurrent with one of the three phases. Local control of the PE is achieved by prohibiting the PE to write to the NNR, RLR, VSR, and the register file and this is controlled by a special purpose counter.

SYMBIC

The SYMBIC system was proposed by Fountain [31] in 1988 for the processing of 512x512 pixel images. The system is designed to optimise both low-level and intermediate-level processing by allowing a certain freedom of local control in each element. The structure of the processing element is shown in Fig 3.12, and each processing element consists of a processing unit and 64 Kbytes of memory, arranged as 32 Kword. Each processing unit incorporates a 16-bit ALU, a 8x8 multiplier (Mult), eight 16-bit registers (B0 to B7), nearest neighbours registers (N0 to N8), bidirectional shift-register (S), an address offset adder (ADDO) and a 16-bit local control register (C). As stated earlier, local controls, including local modification of RAM addresses, local connectivity and PE function, are envisaged. The nearest neighbours registers provide storage space for a pixel's nearest neighbours residing within a 3x3 window. Data residing in the registers can be read selectively thus emulating the effect of modifying the inter-element connectivity. The global broadcast memory address can be altered by the address offset adder. The globally-broadcast control can be modified by the value of the local control register and a switching circuit which enables the bypassing of faulty element is also included in the system. As proposed in 1988, the implementation of the processing unit will be achieved by a semi-custom circuit accommodating one element per chip. The system is targeted to operate at 25MHz.

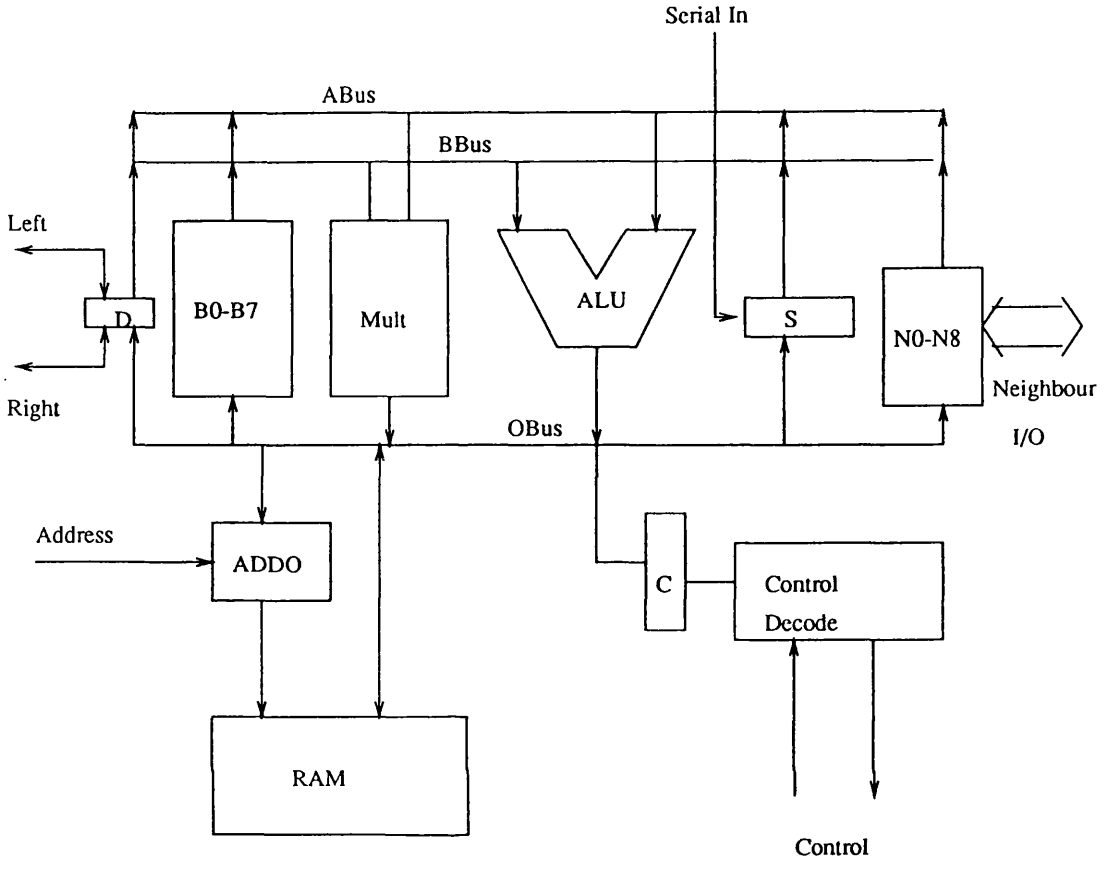


Figure 3.12 Block diagram of a SYMBIC PE

SYMPATI

The Système Multiprocesseur Adapté au Traitment d'Images (SYMPATI) [43] was proposed by Basille et al of Toulouse University in 1981. The system is intended to process 512x512 8-bit pixel images in a wide area of application including remote sensing and biomedical applications. The system embodies two levels of computing elements, the first under MIMD control and the second a SIMD array.

The first level consists of standard processors and memory units as shown in Fig. 3.13(a). The PEs can communicate either through the communication loop or the inter-processor bus. The communication loop is used to transfer a short piece of information and the inter-processor bus is employed for block transfer of data or control signals. The inter-processor bus is linked to the second level PEs via the transfer module controlling the data transfer between the two levels.

The second level consists of 32 processing units each of which comprises of an ALU, a fast scratch pad memory unit, a parallel shift register, and 16 Kbytes of local memory, as shown in Fig. 3.13(b). Each lot of 16 processing units are organised as a group which will process a 512x512 pixel image. The PEs are connected to a fast shifting loop which enables real time video input/output.

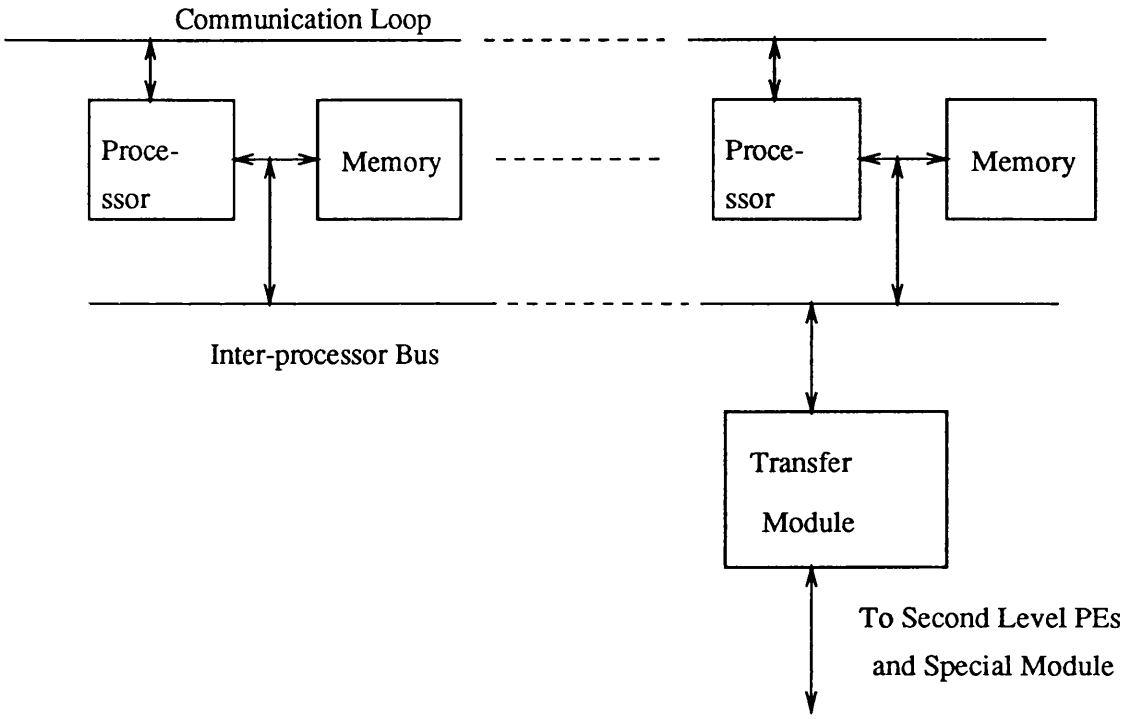
The local memory of each processing unit is used to store 1/16th of a 512x512 image. Each block contains image elements that have the same column number (modulo 16) in an image. The memory can be accessed in two different modes: image scanning and TV scanning. In image scanning mode, successive lines of an image are read, whilst TV scanning mode allows sequential scanning of the odd lines alternating with sequential scanning of the even lines.

Besides the parallel unit, there are special modules, that communicate via the fast bus, performing video I/O, and are controlled by the resource allocator. The system is controlled by microinstructions issued by the command unit and user programmes can be written in a Pascal-like language.

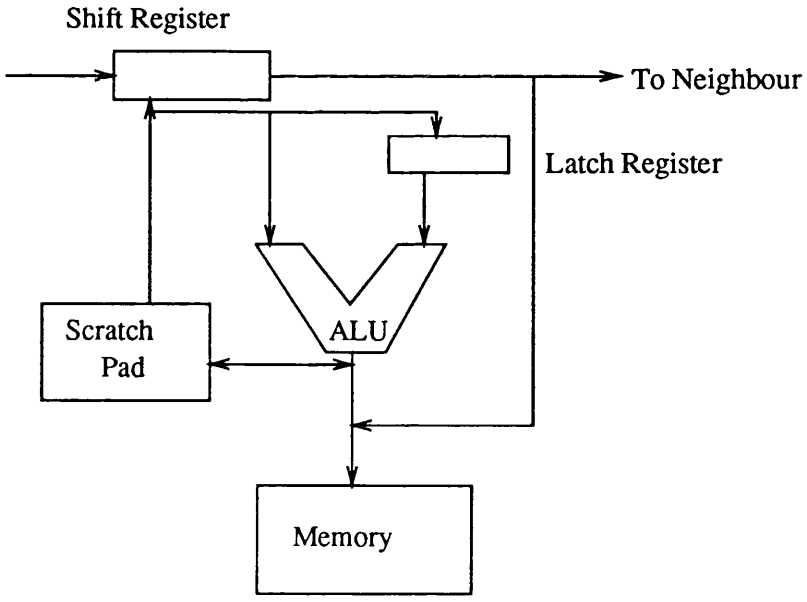
WARP

The WARP [44] computer has been developed at Carnegie Mellon University since 1984. It is a linear systolic array comprising ten processing cells. The major applications of the computer are robot vehicle navigation, scientific computing and signal processing.

The machine is divided into three different units: a host computer; an interface unit (IU) and the WARP processor. The WARP processor consists of ten identical



(a) First level PE of the SYMPATI system



(b) Second level PE of the SYMPATI system

Figure 3.13 Block diagrams of the SYMPATI PEs

cells and each cell can be classified into three functional units, namely the microengine, intercell communication unit and the computing unit. The microengine consists of a microsequencer and a programme memory which can store 8K instructions for the control of the cell. The computing unit, as shown in Fig. 3.14, comprises of a 32-bit floating-point multiplier (MPY), a 32-bit floating-point adder (ADD), 34 Kwords of local memory (MEM), an integer unit (AGU). The components communicate either directly through 16-bit channels or via the multi-input multi-output crossbar network. The local memory is split into a 2 Kword block and a 32 Kword block. The 32 Kword block is used for temporary data storage. The 2 Kword block is used as a backup data storage for the floating-point units. Computation is performed by the floating-point units and the integer unit is mainly used for generating addresses for the local memory.

The inter-cell communication unit (see Fig. 3.14) includes two data buffers (XQ, YQ) and an address buffer (AdrQ), all implemented in First-In-First-Out registers which can store 512 pieces of data. The data buffers accept data, 34-bits wide, coming from a cell's two nearest neighbours. Data coming from the left-hand neighbour is loaded into the XQ buffer, whilst the YQ buffer is provided for data coming from either neighbours. The address buffer, 32-bits wide, holds the local memory addresses and control signals broadcast from the IU. As previously stated, the local memory addresses can also be generated by the AGU and a switching network (Address Cross Bar) is provided to select the source for the local memory address. An integrated version of the WARP cell, known as iWARP, is now under development.

The WARP cell is horizontally microprogrammed using 272-bits wide microcode. Because each cell has its own sequencer and programme memory, multiple instruction multiple data (MIMD) operating mode is achievable. The interface unit handles data input/output between the host and the Warp array and is controlled by a 96-bit wide microcode. The host controls the Warp array and other peripheral and also performs operations not easily mapped onto the array. The host is assisted by two external hosts which are responsible for data transfer to and from the array and other devices. The system can be programmed by the W2 language, which is Algol-like, and it hides all low-level details of the machine from the user.

ZMOB

The ZMOB [45] system is a MIMD array, developed at the University of Maryland since 1979. The system consists of 256 processing elements and 257 mail stops (see Fig. 3.15) that are responsible for the communication between PEs and with the

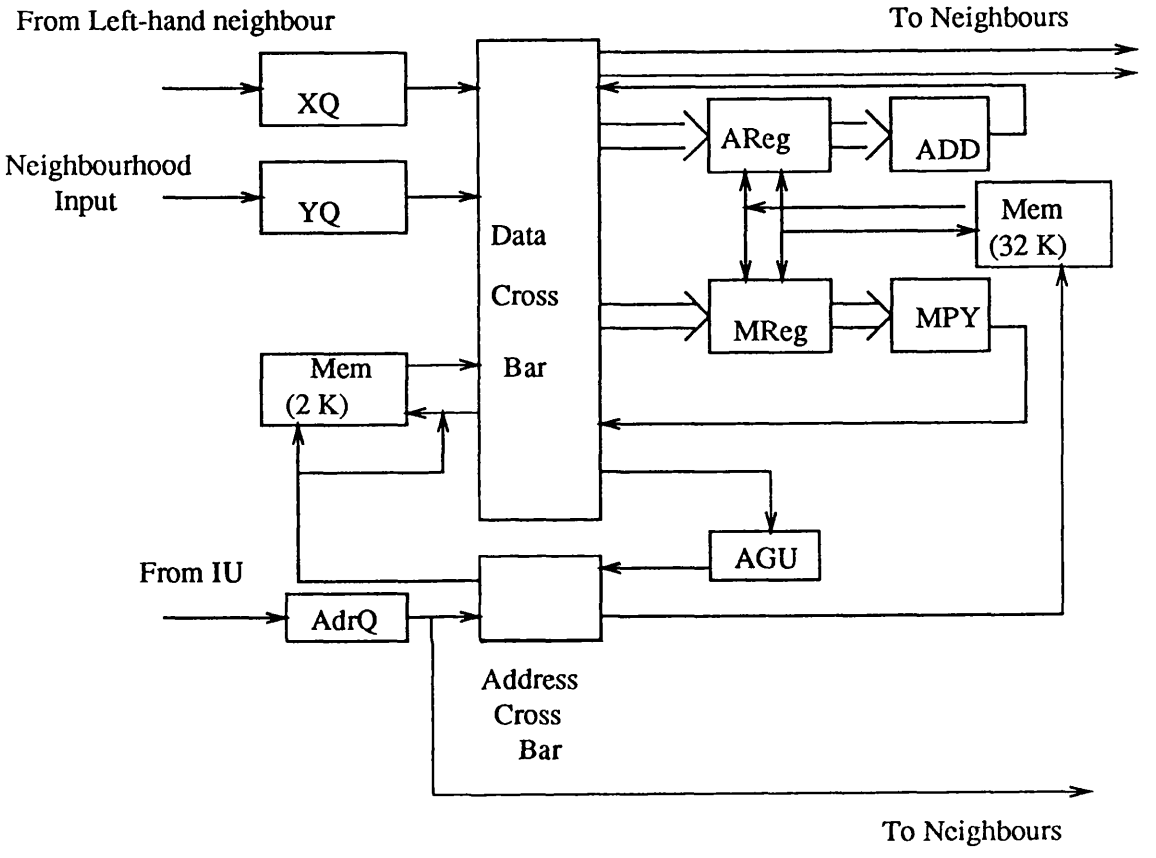


Figure 3.14 Block diagram of a WARP computing unit

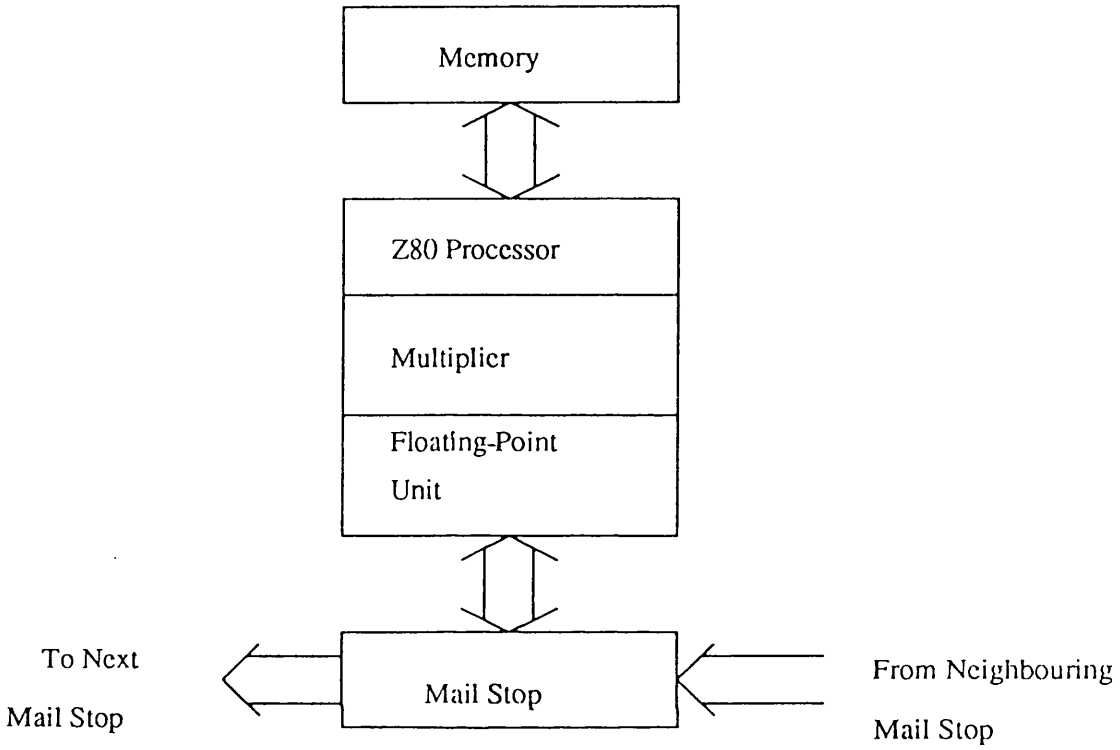
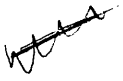


Figure 3.15 Block diagram of a ZMOB PE

external world. Each PE consists of a Z80A microprocessor, 63K of RAM, 1K of EPROM, a 8x8 bit multiplier, a 32-bit floating-point processor unit, 19.2K baud synchronous/asynchronous external serial interface, a 24-bit high speed external parallel interface and an interface to the mail stop. Each mail stop incorporates a 48-bit shift register and 15 special purpose registers. All the shift registers are connected together and form a ring conveying data between PEs and to/from external devices. Each PE is associated with one mail stop and the 257th mail stop interfaces with the external world. The 15 special purpose registers occupy 16 memory locations of the associated PE and reading from, or writing to, the mail stop is via these locations. Communication between PEs is by message passing and a message is a 48-bit data divided into a 8-bit control field, a 12-bit source field, a 12-bit destination field and a 16-bit data field. The destination field can be used to specify either a mail stop address or a pattern. Because of this property, four message transfer modes: send to processor by address; send to processor by pattern; send to all processors and send to set of processors by pattern are possible. Upon receiving a message, a mail stop will examine the destination field and decide whether to accept or reject the message.

In addition to the mail stops, is a serial channel and a parallel channel, both of which are connected to the external world. Such an arrangement permits ZMOB to be used in applications requiring remote control or message switching among remote external hosts. The ZMOB will support various programming languages varying from the Z80 assembly language to LISP, an artificial intelligence language.



3.4 Conclusion

The linear arrays presented in Section 3.3 are summarised in Table 3.1. Each system embodies certain special feature(s) to render it unique. Table 3.1, reflects the diversity in the design of linear arrays. Linear arrays come in different degrees of complexity as regards the computing power of the processing element; communication methodology; controlling mechanism and implementation of the system. This demonstrates the flexibility in the physical implementation of a linear array. The computing units employed by different systems vary from a single bit ALU to a 32-bit floating-point device. Most systems provide 8-bit or higher resolution. The systems described come in any length, from one-PE-per-image-column to a smaller size of only 8 PEs. Considering the number of elements employed, only the PICAP3 system, having 64 PEs, can be regarded as of intermediate size. Both off-the-shelf components and custom-designed devices are used in different systems. The advance in VLSI fabrication technology provides a user with a vast selection of components and this

System	PE	Processing Power	Local Memory	Implementation	Control	Programming
AIS5000	one per column	1-bit	32 Kbits	Custom	SIMD	High-level
ASP		1-bit	64 bits	Custom	MIMSIMD	High-level
CLIP7A	256	16-bit	64 Kbytes	Custom	SIMD	High-level
LAP	256	1-bit	256 bits	Standard	SIMD	
PICAP3	64	16-bit & 16x16 multiplier	4 Mbytes	Standard	SIMD	Microprogram
PIP	8	32-bit	16 Kbytes	Standard	SIMD	
SLAP	one per column	20-bit		Custom	SIMD	
SYMBIC	512	16-bit & 8x8 multiplier	64 Kbytes	Semi-custom	SIMD	
SYMPATIC	16	8-bit	16 Kbytes	Standard	SIMD	High-level
WARP	10	32-bit FPU (adder & multiplier)	32 Kwords	Standard	MIMD	High-level
ZMOB	256	32-bit FPU & 8x8 multiplier	64 Kbytes	Standard	MIMD	High-level

Table 3.1 Physical features of different linear arrays

makes the optimisation of a design easier. The controlling and communication mechanism employed by various systems reflect their different applications. For example, the WARP, ZMOB, and ASP are systems devised for general purpose applications and in those systems an MIMD, or multi-SIMD, controlling mechanism is used and complex inter-element communication methodology is provided. The linear systems studied in Section 3.3 embody various degrees of autonomy, the ZMOB system, for example, includes local sequencing control in its elements, whilst the elements of the PICAP3 have local data control.

This survey only provides a background study to the physical properties of linear arrays but the major issue, concerning the development of a multi-linear-array system, remains unresolved. Because a linear array is the basic architectural unit of the proposed multi-layer system, it is important first to derive the design criteria for an effective linear array. As local autonomy can enhance the performance of a linear array, it should also be studied. The next step towards the development of a multi-layer system is to investigate architectural requirements which are governed by the structure of the chosen image processing problems. There is no simple answer to all these questions except the examination of the problems from practical experience. In order to achieve this, a linear array is selected as the model system.

In the next chapters, the CLIP7A system is chosen as a sample device and different algorithms will be implemented. The exercise is designed to find answers for the above questions. This in turn will provide useful information for the design of a multi-layer system. The investigation will concentrate on four aspects.

1. The effectiveness of local autonomy;
2. The structural properties of a linear array;
3. The processing requirements at different levels of operations; and
4. The relationships between the structure of a problem and the architecture of the system.

The CLIP7A system is chosen because:

1. The CLIP7A system is fully functioning in both hardware and software so application programmes can be developed with comparative ease.
2. The CLIP7A system embodies local data control which provides a starting point for the investigation of local autonomy.

3. The CLIP7A system is available at University College, London.

In the next Chapter, a thorough description of the CLIP7A system is provided which will help the readers to understand the algorithms described in chapters 5 and 6.

Chapter 4
The CLIP7A System

4.1 Introduction

It was concluded in Chapter 3 that the linear array could be employed as the basic structure for a multi-layer system. This is because the linear array can be adopted for different levels of image processing; it is easier to construct, to expand and to achieve fault tolerance. The survey in Chapter 3 Section 3.3 showed the diversity in the design of linear arrays. For the development of a multi-linear-array system, it is necessary to derive the design criteria for an effective linear array. Local autonomy is also an important factor.

Applying the system to solve different image processing problems will provide the relevant information for the design of linear arrays. As stated in Chapter 3 Section 3.4 the CLIP7A system was chosen as an experimental system on which further investigation would be carried out. A detailed description of the system will assist the understanding of the philosophy behind the implementation of the image processing algorithms describing in the coming chapters. In the following sections both the CLIP7 chip and the CLIP7A system will be described.

4.2 The CLIP7 Chip

The design of the CLIP7 chip [46] was based principally on the experience gained from the CLIP4 image processor which is a 96x96 mesh-connected array. The major design parameters for the chip were as follows:

1. A single processor to be integrated to each chip;
2. Major part of local storage to be provided by off-chip RAM;
3. Special arrangements for neighbourhood connections;
4. A limited degree of autonomy to be included on the chip; and
5. A Multi-bit ALU.

The CLIP7 chip is a custom-designed circuit, running at 5MHz, which is implemented in 5µm CMOS technology. Its physical characteristics are given in Table 4.1. Generally speaking the CLIP7 processor is a two bus structure as depicted in Fig. 4.1. The two data busses are the Ibus (for input) and the Obus (for output). Both busses are 16-bits wide. The Ibus conveys data from other components to the ALU, whilst the Obus is used for inputting, and outputting, data from, and to, the external world. Major components in the chip include a 16-bit ALU, a binary gate (BG) and registers

	CLIP7
Technology	5μm CMOS
Size	25 sq mm
Number of transistors	8000
Power consumption	<0.1W
Clock rate	5 MHz
Package	68-pin
Word length	16-bit

Table 4.1 Characteristics of the CLIP7 Chip

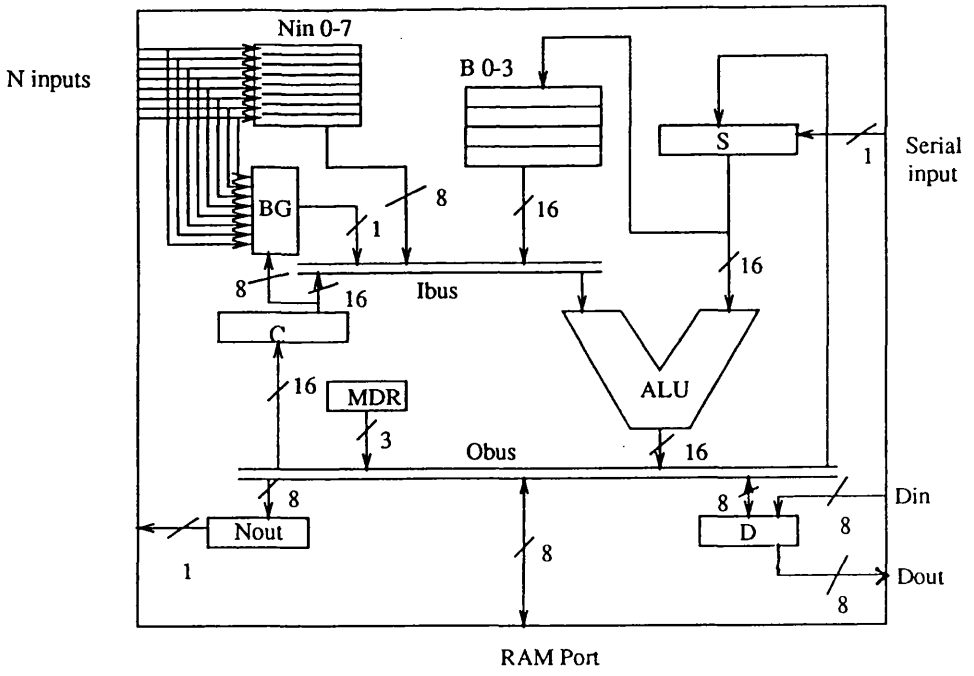


Figure 4.1 The structure of the CLIP7 chip

which are used for various purposes.

The ALU can perform 16-bit addition and subtraction plus all the Boolean operations to its two 16-bit inputs. The ALU gets its input either from the S-register or from the Ibus, whilst its output can be obtained from the Obus. The register set includes a set of four general purpose registers (B-regs); a shift register (S-reg); a set of eight neighbourhood registers (Nin 0 to Nin 7); a neighbourhood output register (Nout); a condition register (C-reg); a data register (D-reg) and the multiplexer direction register (MDR). The D-reg, Nin registers and Nout register are 8-bits wide, whilst the others, excepting the MDR, are 16-bit. Because the functions performed by these registers and the binary gate are important to the understanding of the operations of the system, a brief description of them is given below:

The Binary gate

The binary gate is a special circuit, as shown in Fig. 4.2, which is devised to speed up binary neighbourhood operations. There are two sets of inputs to the binary gate: the neighbourhood inputs and an 8-bit mask pattern (or control signals). The neighbourhood inputs are extracted from the least significant bits of the eight Nin registers and the mask pattern is programmable. The neighbourhood inputs can be selected through the mask pattern and the overall logical 'OR' of selected inputs is provided. The binary gate can be visualised as a binary neighbourhood pattern matcher and can be applied to different binary operations such as skeletonisation.

The B-registers

There are four B-registers, labelled from 0 to 3, which are used mainly as a scratch pad. The B-registers can be loaded through the S-register and their output can be obtained from the Ibus. The read/write to a particular B-register is determined by a 2-bit address which comes either from the C-register or from external control.

The S-register

The value stored in the S-register is taken as an input to the ALU or to the B-registers and it can perform six different rotation and shift operations as shown in Fig. 4.3. The shift function is vital for the processor to perform 16-bit computation. As the RAM port is only 8-bits wide, 16-bit arithmetic, if required, necessitates the following operations to form a 16-bit datum:

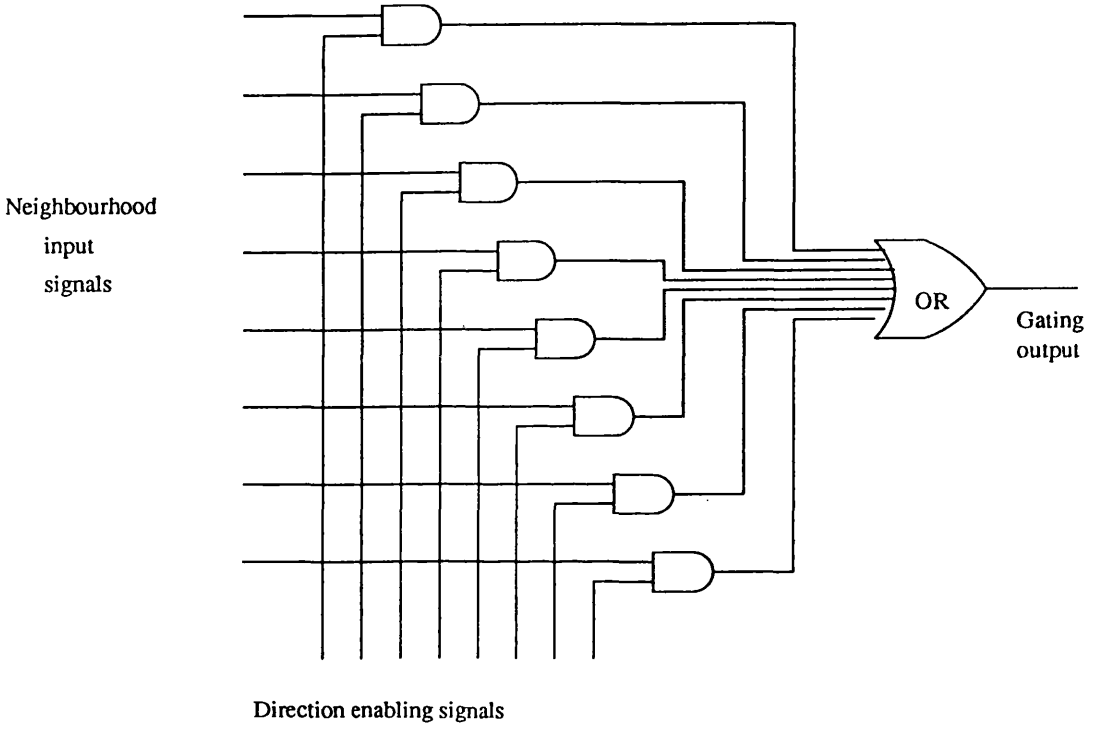


Figure 4.2 The structure of the binary gate

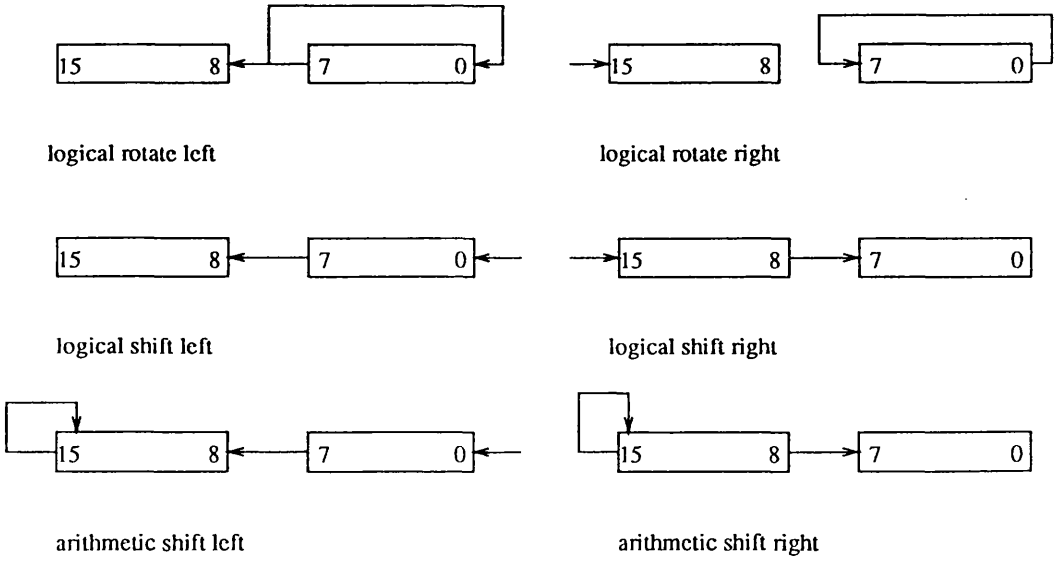


Figure 4.3 Different shifting modes of the S-register

1. Read the high-byte from RAM
2. Load the S-register with the high-byte
3. Shift the S-register 8 times
4. Load the B-register with the S-register
5. Read the low-byte from RAM
6. Load the S-register with the low-byte
7. Combine the high-byte (in B-register) and the low-byte by a logical 'OR'

The S-register can be loaded bit-parallel from the Obus or, alternatively, there is a serial input which allows the serial loading of data into the S register from the controller.

The Neighbourhood registers

There are two types of neighbourhood register: the neighbourhood input register Nin and the neighbourhood output register Nout. There are eight Nin registers, labelled from 0 to 7, which are serial to parallel registers. The Nin registers provide storage for data coming from a 3x3 window. The input of the Nin registers is connected to the Nout registers (a parallel to serial register) and edge registers (see Section 4.3.1) from a PE's two adjacent neighbours. In order to load the Nin registers, data are first loaded into the edge registers and Nout registers. The edge registers, Nin and Nout registers are shifted synchronously. It takes eight clock cycles to complete the data transfer. Data stored in the Nin registers can be read via the Ibus by specifying the Nin register's label represented by 3 bits in the condition register.

The Condition register (C-register)

The condition register can be used as a general purpose register and get its data from the Obus. More importantly it provides conditional control over five areas of processor operations: Nin registers address; binary gate direction; B registers address; ALU functions and conditional load. Further discussion of the autonomous functions of the chip will be included in Section 4.3.

Multiplexer direction register (MDR)

The MDR is a 3-bit register intended for local storage of the direction from which the neighbourhood register will be addressed. It can be loaded externally by the controller or internally from the C-register. The MDR can be used as the source for the C-register's three least significant bits, which specify an Nin register's label.

The D-register

The D-register is a shift register and is devised for data I/O operations. The register can be written to, or read from, the Obus. There are external input and output lines which enable D-registers in neighbouring processors to be connected together to form a high speed data channel.

4.3 The CLIP7A System

The physical orangisation of the CLIP7A system is depicted in Fig. 4.4. It includes a host, video I/O devices, the linear array and the controlling unit. The system is currently hosted by a SUN3 workstation which controls the array through a VME-bus interface. The video I/O device is a Computer Recognition Systems CRS4000-series unit, providing video input and output channels and frame stores, which is interfaced to the host through the VME-bus. The discussion of the CLIP7A system will concentrate on the linear array, the system controller, and the system software.

4.3.1 The CLIP7A Array

The CLIP7A array consists of 256 processing elements, each of which is connected to its two nearest neighbours. The structure of a PE, as shown in Fig. 3.7, consists of two CLIP7 chips, 64 Kbytes of RAM and auxiliary logic.

Communication between PEs is via two uni-directional data channels, called the D-chains, which are 8-bits wide. A D-chain is created by connecting the D-registers of the CLIP7 chips together, so that data can be shifted from one D-register to another. The D-chain formed by the co-processors' D-registers carries data from right to left while the processor's D-chain conveys data in the opposite direction. The D-chains are connected by two data switches at both ends and two different configurations, *reflect* and *wrap-around,* for the data path are possible, as depicted in Fig. 4.5. The configurations are controlled by software and it is possible to reconfigure the connection at run time. Additionally, the D-chains are used for inputting data from, and outputting data to, the controller. The configurations of the D-chains during data I/O are depicted in Fig. 4.6. Data can be input to the array two bytes at a time (to opposite ends of the structure) from an interface (the DBus register in the controller), or output in a similar fashion.

Besides the D-chains, there is another method to input/output data to or from the array and this involves a decoding/encoding network. The network enables the controller to read, or write, bit-serially from, or to, the S-register of a selected PE. The

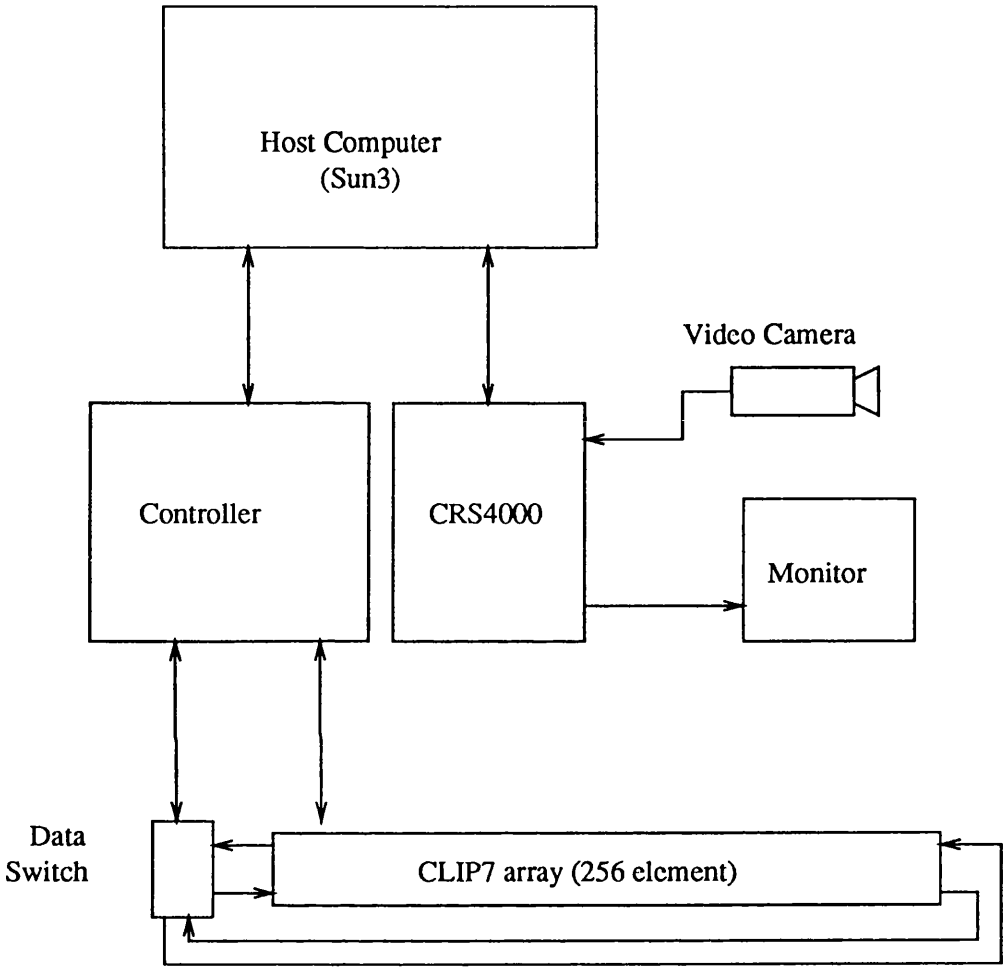


Figure 4.4 The CLIP7A system

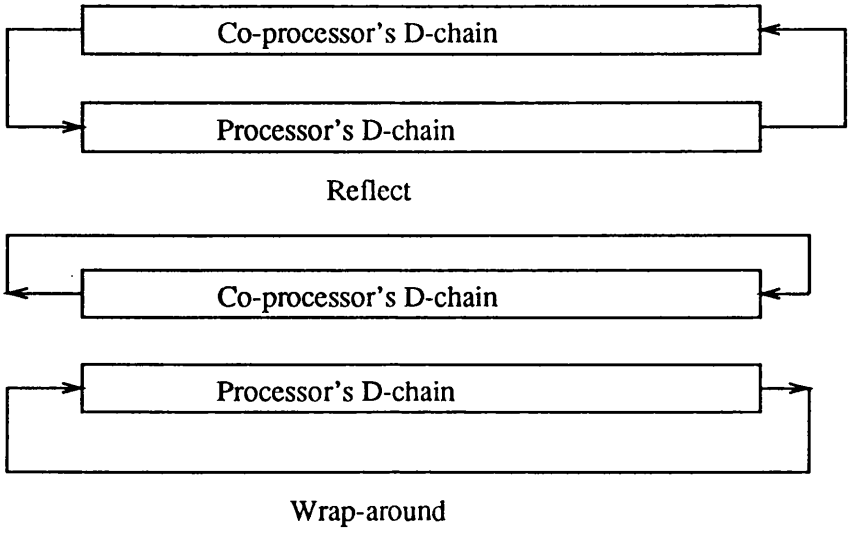


Figure 4.5 Different connection modes of the D-chains

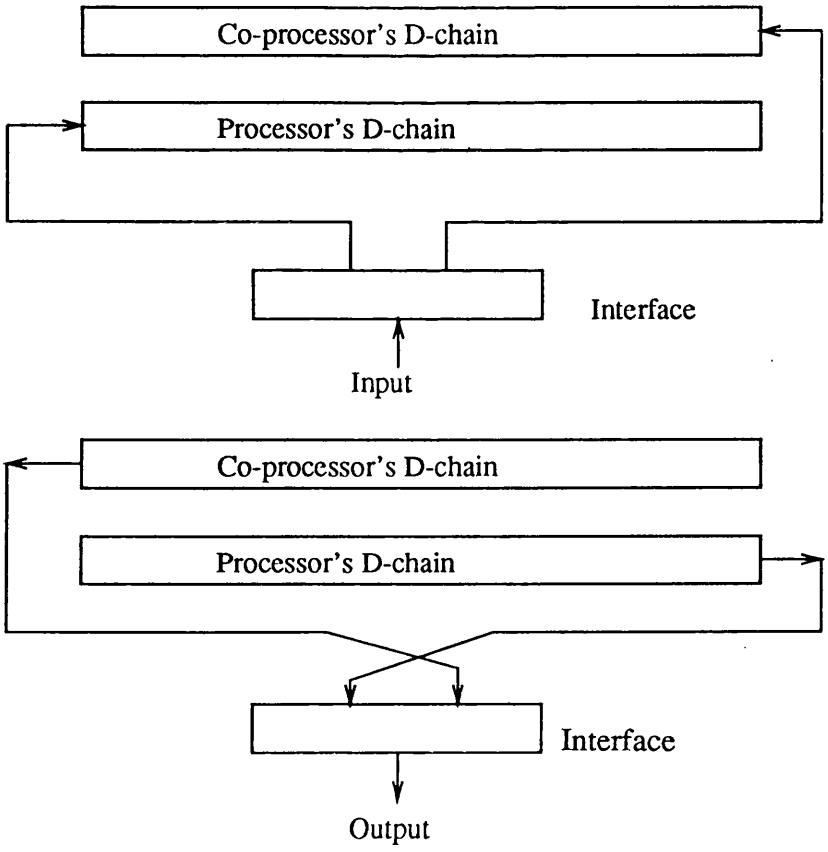


Figure 4.6 Configurations of the D-chains for data I/O

decoding part of the network allows the controller to communicate with a particular PE by specifying its position, from 0 to 255, in the array. The encoding network generates an integer which is equivalent to the position of the first element of the array which has a non-zero value stored in the Nout register. The encoding network output can be used as a feedback signal in different situations.

The complete array embodies a total of 16 Mbytes of memory, which enables the system to accommodate up to 256, 256x256 8-bit images. The instruction stream is broadcast from a single controller but individual PEs can perform autonomous functions. The functions are: conditional control of the CLIP7 processor's operations, local addressing and local neighbourhood connectivity.

The complete CLIP7A system excluding the host and video units is housed in two 6ft high, 19in rack units. There are 128 processor cards each supporting two PEs, 10 control buffer cards and two data switching cards at the end of the array. The 140 cards are housed in five cardframes in the double extended eurocard configuration. The connections between cards are via both backplanes and flat cables.

Conditional Control

The conditional control is determined by the value stored in the condition register and the 16-bit data can control five areas of processor operations: Nin register address; binary gate pattern; B register address; ALU functions and conditional load. The functions of individual bits are summarised in Table 4.2. The first 3 bits can be used to address one of the Nin register, or by combining with another 5 bits, forming the least significant byte, they can specify the mask pattern for the binary gate. Bits 8 and 9 are used for the addresses of the B registers. Bit 10 of the C-register can determine conditional operations of the ALU. If this bit is clear and a conditional operation is selected (through an external control signal **usecc**), the S-register input of the ALU is forced to zero, and the ALU will only operate on its input from the Ibus. This allows the processor to perform the conditional additions required during multiplication.

Bits 12 and 13 define the carry input to the ALU. If bit 12 is set then the state of bit 2 in the ALU's output is used as the carry input. When bit 13 is set then the previous carry output is fed into the carry input. The most significant bit, when it is clear, can disable the loading of three of the on-chip registers, namely the S-register, the Nout register and the C-register itself. The bit is effective only when the external control **usecc** is selected. It can be loaded from one of the ALU status flags - sign, carry, zero and overflow.

Bit Number	C-register Use
0	BG dirn.[0] or N-input address [0]
1	BG dirn.[1] or N-input address [1]
2	BG dirn.[2] or N-input address [2]
3	Binary gate direction [3]
4	Binary gate direction [4]
5	Binary gate direction [5]
6	Binary gate direction [6]
7	Binary gate direction [7]
8	B-register address [0]
9	B-register address [1]
10	ALU arithmetic function control
11	ALU carry input source 3 (round)
12	ALU carry input source 4 (carry)
13	Unused
14	Unused
15	Conditional load

Table 4.2 Bit assignments for the Condition Register

Because the C-register can be loaded directly from the Obus, which is connected to the external world and the output of the ALU, so different values can be loaded to registers in the array. Each CLIP7 chip performs according to the value of its condition register so local conditional control can be achieved.

Local Addressing

As briefly described in Section 3.2, each PE is capable of generating a half, or full, address for its local memory. The CLIP7 chip has no address bus, however, the chip's 8-bit data bus can be employed as an address source and this has been applied to the CLIP7A system design. Each processing element consists of two CLIP7 chips, labelled processor and co-processor respectively. The co-processor is responsible for generating local memory addresses from its 8-bit data bus. To access the 64 Kbyte RAM a 16-bit address is required but there is only a 8-bit address bus within the PE and therefore multiplexing is necessary in order to address various locations in the memory. A 16-bit address is divided into a high-byte and a low-byte. The high-byte address is generated first and is latched into a 8-bit register. Once the high-byte is stored the low-byte address will be generated and occupy the address bus so that a full 16-bit address is created. The source for both bytes can come either from the co-processor's data bus or be globally broadcast from the controller and three kinds of addressing modes can be achieved, as shown in Fig. 4.7. The high byte or low byte value coming from the co-processor can be data stored in the local memory or a result obtained from previous processor operation. Due to the local addressing property, each PE can address a different location of the local memory at the same operation cycle and the implementation of lookup tables is an example application which is described in Chapter 5.

Neighbourhood Connectivity

In a linear array each element is physically connected only to its two nearest neighbours, East (right) and West (left), and the CLIP7A system embodies special hardware to emulate the connectivity (between a PE and its eight nearest neighbours) employed in a 2-D array. Data belonging to a PE's north and south neighbours (in a 2-D array) are stored in the local memory of a CLIP7A PE. Each CLIP7A PE includes two edge registers which are employed to hold these data temporarily. Once the edge registers are loaded with data, a PE can access all the data within a 3x3 window through them and the Nout registers from the neighbouring elements, as shown in Fig. 4.8. The edge registers are parallel-to-serial registers and their output together with a

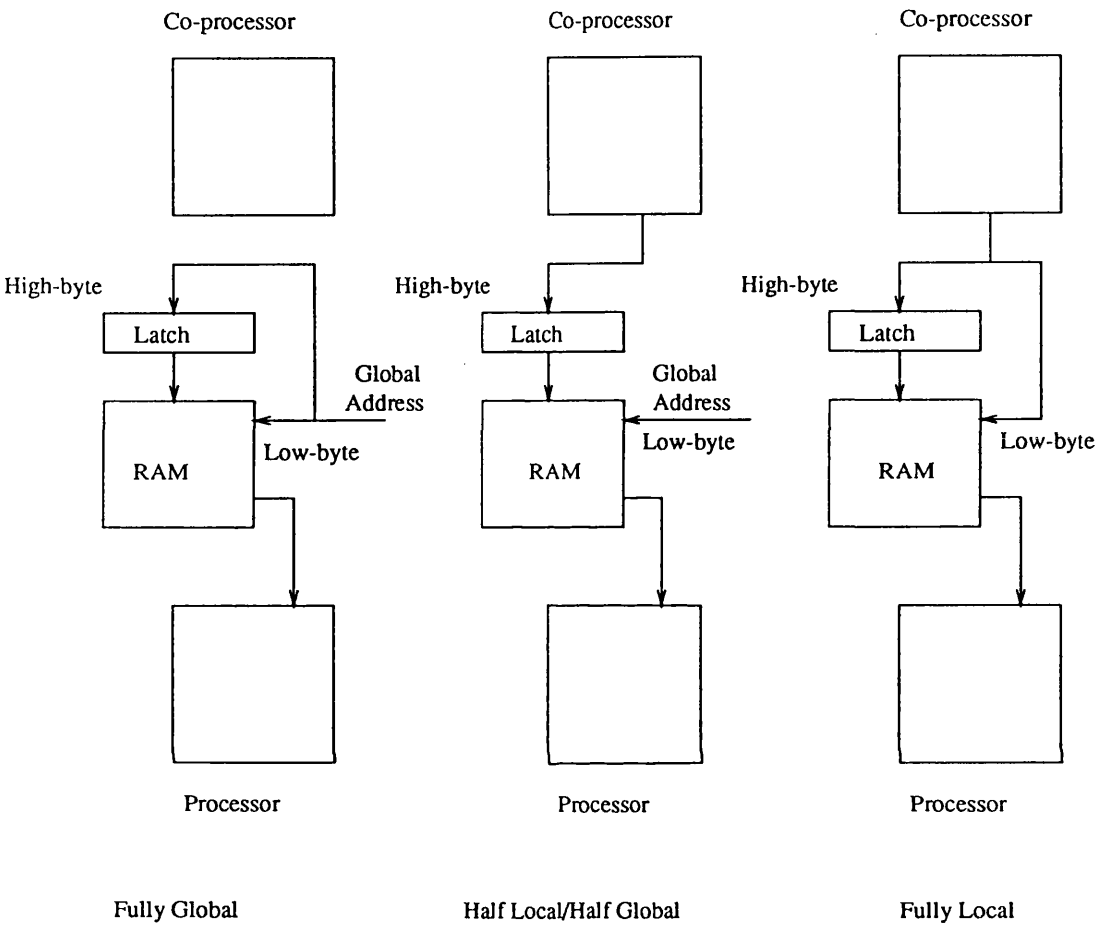


Figure 4.7 Different addressing modes of the CLIP7A PE

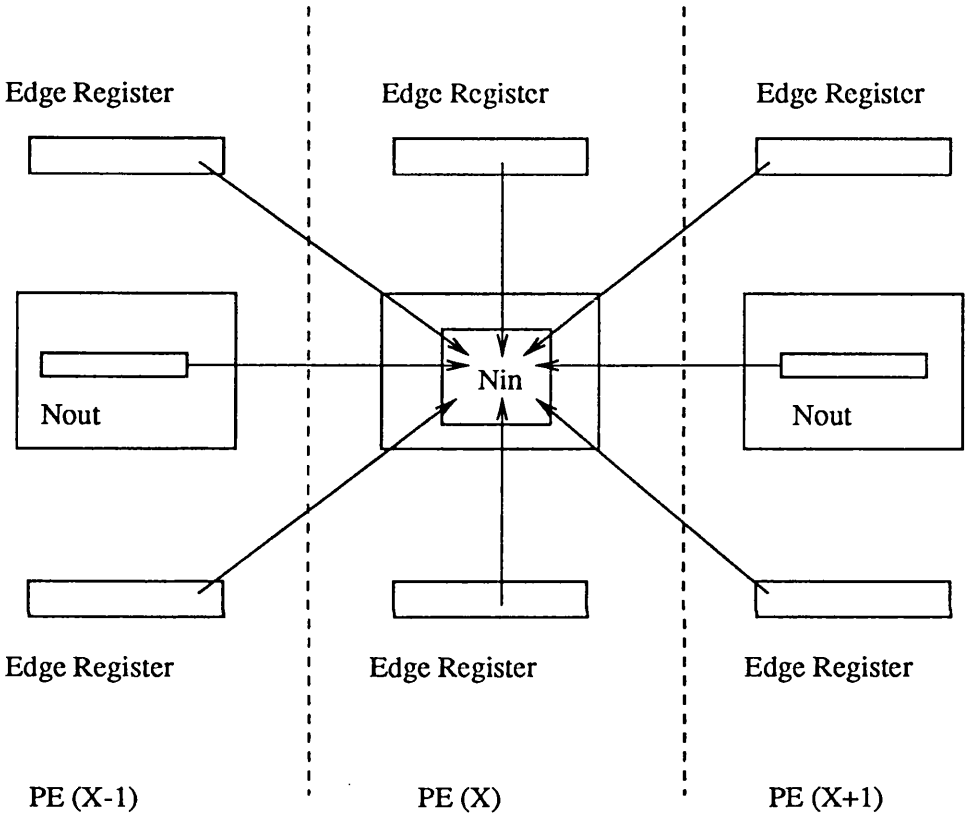


Figure 4.8 Emulation of the nearest neighbours connectivity.

PE's two real neighbours' Nout registers are connected to the Nin registers of PE(x), as shown in Fig. 4.8. Data is shifted bit-serially into the Nin registers in parallel. Once the data are stored in the Nin registers it is possible to extract them from a particular direction (see Fig. 3.3) by specifying the proper Nin register number in the condition register. So data lying within a 3x3 window can be addressed independently through the Nin registers and this emulates the establishing of a communication link between a PE and one of its 8 nearest neighbours as in a 2-D array. Because each PE can address different Nin registers at the same instant, the result is similar to each PE being connected to a different neighbour.

4.3.2 System Controller

The CLIP7A system is microcode controlled and horizontal microprogramming is employed. Each micro-instruction consists of 160 control signals which operate both the linear array and the controller. The system controller consists of a microcode sequencer and a microcode programme store. The controller, however, also acts as an interface between the linear array and the host computer. The system controller occupies six locations in the host's VME bus address field. Two locations are employed for data input/output and the remaining four are used for control and feedback signals. The structure of the controller is depicted in Fig. 4.9. The controller can be divided into three functional units: an interface unit, a computing unit and a microengine.

The Interface Unit

The interface unit enables the controller to communicate with the host and the array. The unit includes two 16-bit pipelined data channels called the TO-pipe and the FROM-pipe. The channels are uni-directional and the host will only write to the TO-pipe and read from the FROM-pipe. In the original design the two pipes are implemented by 16 word deep First-In-First-Out registers. Each pipe comprises of 2 registers arranged in series so that up to 32 words of data can be stored. Data passes through both registers before reaching its destination (either to the array or the host). The registers in a pipe must be synchronised so that data can be transferred from one register to the other. In the original circuit, there was a problem in synchronising the registers and data were lost when moving from one register to another. The problem was resolved by a new design employing a single device that can store up to 512 bytes of data. The new design also provides more data storage so that either the array, or the host, can insert more data to the pipes before they are full.

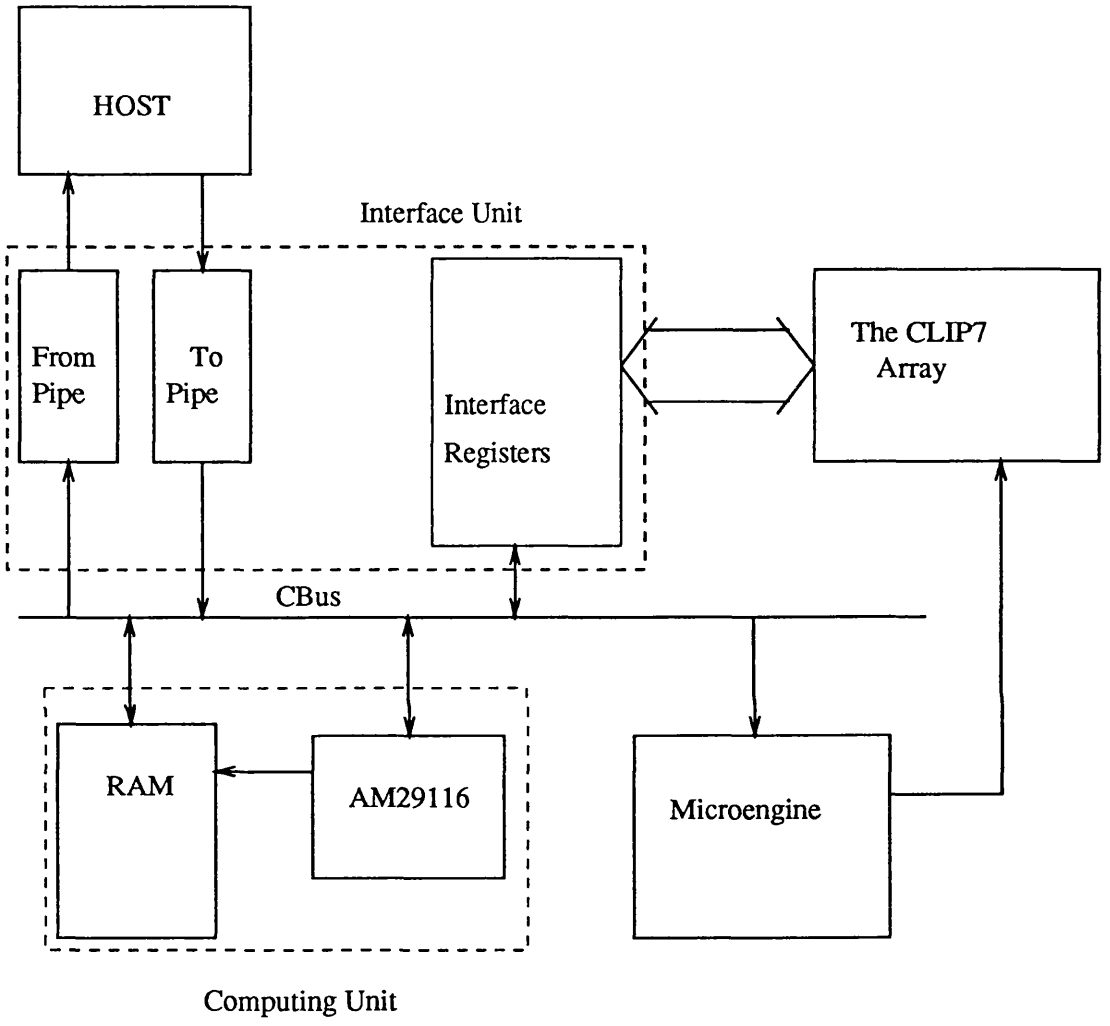


Figure 4.9 Block diagram of the CLIP7A controller

Communication between the controller and the array is through five 16-bit registers. The registers comprise: a data register, Dbus register, serial register, position register and connect register, and their functions are:

Data: for sending RAM addresses to the array. The output of the register is connected to the co-processor's data bus. At each operation only 8-bits are transferred because the address bus of the PE is 8-bit. A control signal determines whether it is the high byte or the low byte of the register that is broadcast.

DBus: for sending data to, or receiving data from the array. The high-byte of the register is connected to the processor's D-chain, whilst the low-byte is connected to the co-processor's D-chain.

Serial: for serial data transfers between the array and the controller. Serial data input to this register from the array is derived from the encoding network, whilst the output is used together with the Position register to send data to a selected PE in the array.

Position: used in conjunction with the serial register. It can be loaded either by the host or the linear array. The value loaded by the array is the position of the first non-zero processor (determined according to the data stored in a PE's Nout register). An address written by the host specifies the location of the PE for inputting data serially from the Serial register.

Connect: the value of this register will define the connectivity of the array. It controls the Dbus edge connectivity and globally defines the value of the label for the Nin register.

The Microengine

The microengine includes a microcode sequencer (an AM29331) and a microcode programme store of 16Kx160 bits. The programme store is loaded during system initialisation and the microcodes control the operations of the array, the computing unit and the microengine itself. Operations of the sequencer are controlled by 64 instructions including conditional branch and looping.

The Computing Unit

The Computing unit consists of an AM29116 processor, which is a 16-bit microprogrammable processor, and 64 Kwords of memory. Because the system is a linear array only a single row of an image is processed at each operation. In order to process the entire image a single operation, or a group of operations, must be repeated, proportional to the size of an image. As the instruction stream comes from the host it implies that the rate of communication between the host and the controller must be very high. The function of the computing unit is first to store the instruction stream which is going to be repeated in the coming operation in its RAM. The instruction stream instead of being issued by the host now comes from the computing unit thus reducing the demand for a high communication rate between the host and the controller. This arrangement improves the system's performance.

4.3.3 System Software

The software developed for the CLIP7A system is a hierarchical structure which can be represented graphically as shown in Fig. 4.10. The lowest level is the microcode. As described earlier there are 160 control signals which are used to control the linear array as well as the array controller. An operation, either performed by the array or the controller, is controlled by a group of signals and all the operations that can be performed are represented in the form of mnemonics. The information relating to the mnemonics and the control lines is provided in two definition tables. Microcode programmes can be written in the mnemonics and then translated into the control signals by the Mik compiler [47]. The Mik compiler can be regarded as an assembler, whilst the mnemonic is analogous to the system's instruction set. The control of the system is by horizontal microprogramming and a single micro instruction can invoke a number of operations simultaneously. This feature is supported by the Mik compiler by including two basic constructs, parallel and sequential. For the parallel construct, two types of syntax are accepted and the first one takes the form:

```
load_b;b[1];carry_zero;load_ram
```

In the above example, three operations: loading the b[1] register, loading the ram_port, and clearing the carry flag, will be executed in parallel. Processes written on a single line, separated by semi-colons, will be performed in parallel. The second format employs a symbol, PAR, to mark the parallel operations, this is shown in the following example.

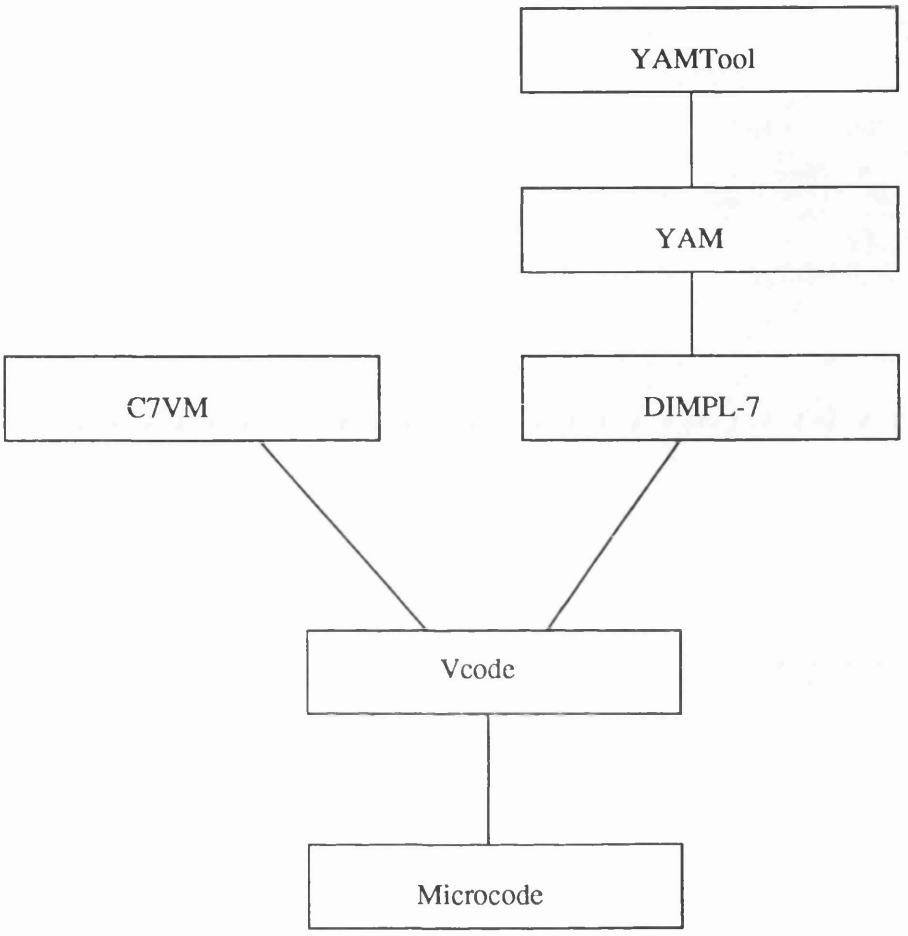


Figure 4.10 An overview of the CLIP7A system software

```
PAR
    load_b;b[1]
    carry_zero
    load_ram
```

The indentation after PAR is required to delimit the action of the parallel construct.

Two forms of sequential syntax are possible similar to the parallel construct.

The first one is:

```
load_b;b[1]
carry_zero
load_ram
```

The three operations are performed in sequence now. The second format uses the symbol SEQ to delimit the construct and this allows the nesting of parallel and sequential processes, as shown below:

```
PAR
    SEQ
        operation1
        operation2
    SEQ
        operation3
        operation4
```

The two sequential constructs will now execute in parallel, i.e. operation1 and operation3 will be carried out at the same cycle and then followed by operation2 and operation4. The Mik compiler also supports procedures which are devised to perform particular functions. A procedure is identified by a preceding *proc* statement followed by the procedure name and the operands of the procedure within brackets, as shown below:

```
proc example()
    load_b;b[1]
    carry_zero;load_ram
```

The parallel construct can also be applied to procedures, it adopts the format:

```
PAR
        example()
        example1()
```

where both example(), and example1() are user defined procedures which are proceeding simultaneously.

In the last example synchronisation between the two procedures may be required at some points. In Mik, a control flag called *tag* is provided to synchronise concurrent microcode processes. The idea of using tags is explained fully in the following example:

```
proc example(tag)          proc example1(tag)
      :                          :
      :                          :
   <tag>     synchronise      <tag>
```

The tag name has to be incorporated into procedure arguments and it is delimited by the angled brackets within the procedure. The procedures will get synchronised at the point of the tag.

In most assembly languages, label is an very important feature and it is also supported by Mik. A label represents a location in the programme that can be used in different branching operations. In Mik, label is delimited by "!", like !label!.

Programming in the mnemonic requires a very good knowledge of the system hardware and it is desirable to provide an alternative programming methodology for users with a limited knowledge of the system. In order to achieve this a microcode library, called the Vcode [48], was written. The Vcode acts as an interface between the microcode level and any second level programming method. The Vcode can be divided into two components: an interpreter and the subroutine library. The interpreter will examine the instruction issued by the host and then invoke the correct subroutine to carry out the operation. The Vcode subroutines define operations in three major areas: controller instructions, array instructions and addressing modes.

The controller instructions handle the reading and writing of the interface registers described in Section 4.3.2. It also includes the control of the connection of the D-chains. The array instructions contain arithmetic operations, Boolean operations, neighbourhood operations, and data I/O. Both byte and word format are catered for in most operations. Five different addressing modes are supported: these consist of

immediate, direct, indirect, locally indexed global page, and remote direct. The first three are common to all computing languages. The remaining two exploit the special physical features of the system. In the locally indexed global page mode the more significant byte and the less significant byte of an address broadcast from the controller are used separately. The less significant byte is used to address a location on page zero of the local memory and the value stored in that content is read and then used as the low byte of the final address by the operation in progress. The broadcast more significant byte is used as the high byte of the final address. The remote-direct mode enables a PE to read data residing in other PE's local memory by specifying the distance between them and the memory location of the source PE is broadcast from the controller. The Vcode provides a comprehensive set of operations which form the core for the higher level programming methodologies (see Fig. 4.10).

Above the Vcode is the CLIP7 Virtual Machine (C7VM) [37]. C7VM was devised as an intermediate programming language, but it supports the implementation of complex image operations. The language provides a data type Vector which maps into the linear structure of the array. Each vector contains 256 elements which can be either 8-bit, or 16-bit, integers and each PE in the array will operate on a single element in the Vector, so a vector operation is performed in parallel in the linear array. A class of vector operations are available and the format:

vm_aadd(src1, src2, dest1)

is used, where src1, src2, and dest1 represent three vectors and the operation vm_aadd will perform an addition of src1 and src2 with the result stored in dest1. A C7VM operation will invoke the corresponding subroutine included in the Vcode library. Since C7VM only supports vector operations, a user programme must define its own image operations. For example, adding two 256x256 pixels images together will require the execution of the vm_aadd operation 256 times. In addition to the vector operations, C7VM also includes image input and display utilities. The C7VM is an extension of the C++ language [38] so all the special features supported by the C++ language can be applied to the C7VM and this makes programming at this level flexible. The algorithms described in the following chapters are all written in the C7VM language and because of this reason only a brief description of the alternative programming method is given.

In addition to the C7VM, there are the Device Independent IMage Processing Library (DIMPL-7) [49], Yet Another Menu (YAM) [50], and YAMTool [51]. Together they form a different programming approach. The DIMPL-7 library provides a set of image processing routines which exploit the Vcode subroutines. All

routines available in the DIMPL-7 library operate on 256x256 pixel images and they are confined to a format, in terms of parameters included in the routine and the value returned. Because there are two other DIMPL libraries, DIMPL-4 for the CLIP4 system and DIMPL-sun for the Sun workstation, standardising all the DIMPL libraries has enabled the development of a device independent package - YAM and YAMTool. Yam is a device independent image processing interface and YAMTool is the graphics version of it. YAM can be interpreted as an image processing operating system which provides a user image processing facilities at the command level. A YAM command will invoke the corresponding DIMPL-7 routine which defines the operation to be carried out in the CLIP7 array. Within the YAM environment, a user can enter the name of the operation in the command line like:

```
yam> im1 = add(im2, im3)
yam> display(im1)
```

im1, im2, im3 are images, in the first line, im2 and im3 are added together and the result is stored in im1. The second line is to display the image im1.

The major differences between C7VM and YAM are the data types and the basic operations. In both YAM and YAMTool, the image is the basic data type and image operations such as thresholding and shrinking are provided. In C7VM, an image has to be defined as an array of 256 vectors and image operations have to be implemented from the basic instructions, e.g. arithmetic and Boolean operations, provided by the system. If the CLIP7A system is regarded only as a low-level image processor then, from the user's point of view, YAM and YAMTool provide a straightforward software solution. When the CLIP7A system is used to reflect the merit/demerit of a linear array, however, then a programming language which resembles the structure of the system is preferable. Another goal of implementing algorithms in the CLIP7A system is to explore the usefulness of local autonomy. The subroutines available in the DIMPL-7 library may not be exploiting such properties, so a special set of subroutines must be developed. Lastly, the operations provided in YAM cater for low-level processing only and operations for other levels have to be implemented in other programming methodology, such as C7VM. In order to achieve a coherent programming style C7VM was chosen.

4.4 Summary

In the above sections different aspects of the CLIP7A system have been described. One of the objectives of the CLIP7 project is to employ the CLIP7A system as a research tool for the development of an architecture suitable for tackling image processing problems and this coincides with the aim of this thesis.

The design of the CLIP7 chip was based on the experience gained from the CLIP4 image processing system and was initially envisaged as a general purpose processor for computer vision. The chip embodies several special features that will improve the efficiency of performing neighbourhood data transfer, data I/O and also enable local control of the chip.

The CLIP7A array tries to exploit the hardware features provided by the chip. This can be reflected from the emulation of the eight nearest neighbours connectivity and the dual data I/O channels. In addition to these, local autonomy is an interesting area which calls for in depth study. The system can be programmed in two different ways. The first one provides a high-level environment which hides all the hardware properties from a user. The second one can be regarded as an intermediate-level language. By providing the data type Vector, the C7VM provides a better approach for the examination of the properties of a linear array.

This Chapter provides a thorough description of the CLIP7A system, which should help the readers to comprehend the algorithms described in the next chapters. The algorithms are devised to examine issues relating to the development of a multi-linear-array system, as described in the last chapter.

Chapter 5
Image Processing in the CLIP7A System I

5.1 Introduction

In order to develop a multi-linear-array system investigation in two different aspects have to be considered. The first, concerns the design criteria for linear arrays; the second, architectural requirements derived from the image problems, which the envisioned system is trying to solve. From these, four relevant topics are derived and they are:

1. The study of the effectiveness of local autonomy;
2. The study of the structural properties of a linear array;
3. The study of the processing requirement at different levels of operations; and
4. The study of the relationships between the structure of a problem and the architecture of a system.

This Chapter presents the investigation on the first two topics, whilst the remainder are covered in Chapter 6.

Image processing problems related to these topics are derived. Algorithms devised to solve these problems are implemented in the CLIP7A system, which is utilised as a sample linear architecture. The execution time required for a particular algorithm serves as an indicator for the performance of the system. The execution time obtained from an algorithm devised to examine local autonomy, or the structural properties of a linear array, is compared with that of an algorithm using the general approach which refers to algorithms implemented without exploiting local autonomy and the structural properties of the system. Processing time is measured by the function *ftime,* which is supported by the Unix operating system. The mechanism of the timing procedure is analogous to that of a stop-watch and is explained in the following example.

```
ftime (start);      /* start the clock */
      {
      codes for the image processing algorithm
      }
ftime (stop);       /* stop the clock  */
```

Both **start** and **stop** represent units of time as defined in the Unix operating system. **Start** marks the time at the beginning of an image function and **stop** that at the end. The execution time taken by the function can be evaluated by subtracting **start** from

stop. The minimum time unit that the function *ftime* can measure is 1ms. Operations taking less than 1ms to complete must be repeated many times in order to allow sufficient accuracy of measurement.

The CLIP7A system embodies local data control, meaning that each element can select the destination and source for its data during an operation. These can be a memory location in a PE's local memory or a neighbouring element. If the data comes from the local memory then it is regarded as a local addressing operation. Ability to select data from a neighbouring element is regarded as a neighbourhood reconfiguration function. Local addressing and neighbourhood reconfiguration are studied separately. The former is applied to two algorithms which involve the use of lookup tables and the latter emulates the passing of data between elements in a mesh-connected array, the process being applied to an object identification problem.

The remainder of this Chapter is dedicated to a study of the structural properties of a linear array. Image processing in a linear array is carried out in a scanning manner, with the image operation requiring the repetition of a vector operation (see Chapter 4 Section 4.3.3) 256 times. If an image operation applies only to areas of interest in an image then the number of iterations for the vector operation can be reduced and this in turn may reduce the execution time of an operation. In order to benefit from this, a pre-processing algorithm was implemented. The function of the algorithm is to locate the interesting regions in an image so that image operations can be applied only to those regions. Besides the scanning property of a linear array, data transfer between elements of the array is studied via two geometric transform operations, which include reflection and rotation by 90 degrees.

The algorithms described in the following sections are written in the C7VM language (see Chapter 4) and the term *operation* refers to a vector operation as described in the C7VM manual [37].

5.2 Local Data Control

Local data control enables each processing element to select the source, or destination, of the data used by a globally-determined process. There are two types of local data control, namely local addressing and neighbourhood connectivity.

5.2.1 Local Addressing

Local addressing refers to the generation of local memory addresses, which are utilised either as a source, or destination, of data during an operation. One application of local addressing is the implementation of lookup tables. The application of a

lookup table can be represented by Equation 5.1:

$$y=F(x) \tag{5.1}$$

A segment in the local memory is assigned as a lookup table with the values stored in that segment determined by function F. For example, if F is the square function for positive integer then y=16 when x=4. Therefore 16 is stored in the fifth location (assuming the value of x started from 0) of the table. Extracting the value 16 from the lookup table requires the evaluation of the address of the fifth location. The address is equal to (offset+4), where offset is the address of the first element in the table, as shown in Fig. 5.1. When each PE carries the identical lookup table at the same memory locations then squares of elements in the input vector can be obtained by first adding the offset value to the input vector. Elements in the resultant vector are utilised as addresses of the local memory in order to retrieve the final result. If the offset value is a multiple of 256, the size of a table is confined to 256 locations and only 8-bit values are stored in each location, then each location in the table will have the same high-byte address. When more than one lookup table is employed then tables can be identified by the high-byte values of their addresses. Moreover, the value x can now be used directly as the low-byte address and adding the offset value to x is no longer necessary. Employing the half-local and half-global addressing mode, lookup tables can be easily implemented. Each PE carries the identical lookup table(s) at the same memory locations. The globally broadcast high-byte address determines the lookup table and the locally generated low-byte address selects the proper element in the table. In the next two sections, algorithms employing lookup tables to perform multiplication and skeletonisation are described.

5.2.1.1 Multiplication by Lookup Tables

As described in Chapter 4 Section 4.2, the CLIP7 chip's ALU can only perform subtraction, addition and Boolean operations. Multiplication in CLIP7A is carried out by the shift and add technique [52] and such a method takes about 300 clock cycles to perform a 8x8-bit multiplication of two vectors.

The multiplication of two positive integers can be performed by addition of their logarithms, followed by the application of the anti-log to the sum of the addition. According to the information provided in the Vcode manual [48] this will take about 200 clock cycles. From the estimated figures, the log and anti-log algorithm should take only around two-third of the time required by the shift and add method. Based on this information, the log and anti-log algorithm should be more effective. In order to

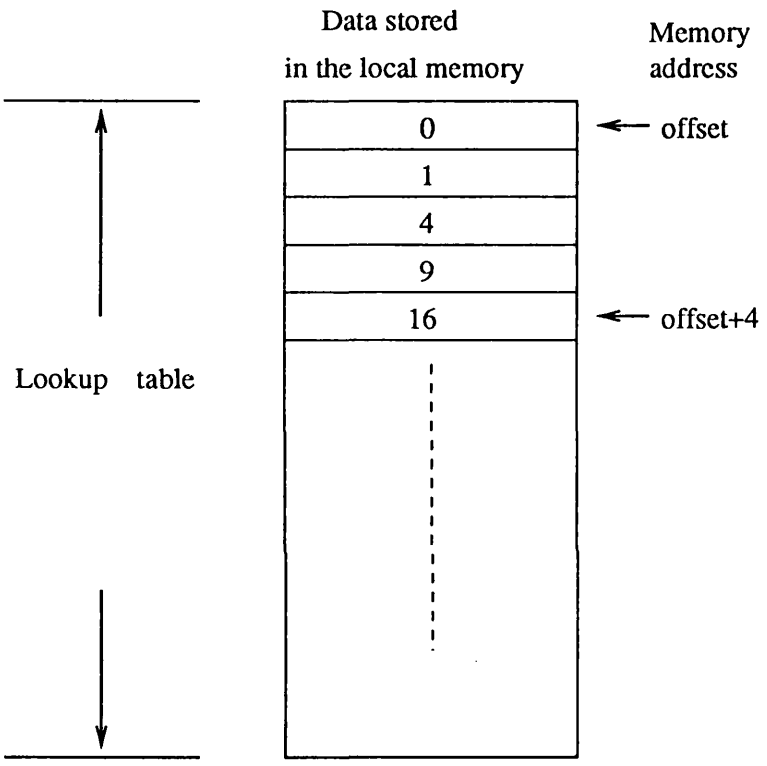


Figure 5.1 The implementation of a lookup table in the local memory

verify this the algorithm was implemented and lookup tables were used to convert the input values to their logarithms or vice versa. Two lookup tables are provided, one of them corresponding to the log-table whilst the other is the anti-log table. Both tables have 256 elements and can be identified from their table numbers. Half local and half global addressing mode is applied and the global address is the lookup table number. In addressing the log-table, the local addresses are the values carried by the input vector, whilst the sum of the logarithms is used to address the anti-log table. Since the local address must be a 8-bit value, this implies the sum of the logarithms must be a 8-bit integer. In order to ensure that, the values stored in the log-table are only 7-bits wide. The function derived to setup the log-table is:

$$Y=16*(log_2(X)) \tag{5.2}$$

X represents integers ranging from 1 to 255

Y represents integers stored in the lookup table

Since the value of Y is represented by a 7-bit data and only integer multiplication is considered in this case, in order to simplify the algorithm, Y represents an integer. When X is 0, a 0 is assigned to Y. The value 16 is a scaling factor which extends the range of Y. It is required because the values of Y are integers. The term $\log_2(X)$ gives a result varying from 0 to 7, without the scaling factor, the values of Y will be integers ranging from 0 to 7 only (this will produce a very low accuracy to the computation). The values of Y evaluated from equation (5.2) vary from 0 to 127 so the sum of any two logarithms is within the 8-bit limit.

The anti-log table is generated from equation (5.2) as well, but this time values of X are stored in the memory locations represented by the values of Y. As an example, when X is 2, Y is 16, so at location 17 of the anti-log table the value 2 is stored. According to equation (5.2), the maximum value of Y is equal to 127 when X is 255 so in the anti-log table value 255 is assigned to locations from 128 to 255.

The computation required to setup the tables is performed by the host and values to be stored in the corresponding memory locations are broadcast to the array. The memory locations included in each table share the same high-byte address.

Excluding the procedure to setup the log/anti-log table, it takes four operations to perform the multiplication of two vectors and they are:

vector LV1 = log(input vector V1);

vector LV2 = log(input vector V2);

vector SUM = LV1 + LV2;

result = anti-log(vector SUM);

Table 5.1 depicts the timing results obtained from the application of the lookup table algorithm to the multiplication of two images compared with those for the shift and add method, which is considered as the standard operation. Setup time for the lookup tables is also included.

	shift and add	lookup table
no. of operations in C7VM level	1	7
256x256 pixels multiply (ms)	29.7	24.1
lookup tables set up time (ms)	0	380

Table 5.1 Execution time for different multiplication algorithms

The lookup table algorithm provides an improvement of about 18.9% over the shift and add method but the set up time for the lookup tables represents a substantial overhead when compared to the execution time of the multiplying function. The shift and add technique is written as a subroutine in the Vcode library (see Chapter 4 Section 4.3.3) and implemented as a single operation in the C7VM level. The lookup table algorithm requires 4 vector operations in the C7VM level, excluding the table set up procedure, meaning that its programme structure is more complex. The accuracy of the algorithm is another disadvantage: the shift and add method generates a 16-bit result but only a 8-bit result is provided in the lookup table algorithm. It is impractical to implement the lookup table algorithm when higher precision is sought, due to the amount of memory required to accommodate the lookup tables. If a final result of 16-bit is needed, for example, then the anti-log table will occupy 128 Kbytes of RAM.

5.2.1.2 Skeletonisation by Lookup Table for Binary Images

The second algorithm employs lookup tables to perform skeletonisation in binary images. Skeletonisation is utilised in different character recognition problems (see Chapter 6) during the pre-processing stage. The skeleton of an object can be obtained by successively removing the outer layer of pixels from shapes, whilst retaining any pixels whose removal would alter connectivity or which are the ends of branches of the skeleton. The process is complete when no further pixels can be removed without altering connectivity or shortening the skeleton.

The Arcelli algorithm [53] works by comparing each pixel and its neighbours with a series of masks as shown in Fig. 5.2. In Fig. 5.2, the zeros are pixels which must have value 0, the ones are pixels which must have value 1, and the values of the remaining pixels are immaterial (or don't care) represented by Xs. The masks are applied in the sequence A1, B1, A2, B2, A3, B3, A4, B4. When the pattern formed by a pixel's eight neighbouring elements matches with a mask's pattern then the pixel is removed. The masks are applied repeatedly until no further pixels are deleted.

In the CLIP7A array, the matching between a mask and a pixel's neighbours is achievable using hardware. The binary gate (see Chapter 4 Section 4.2) is the device, embedded in the CLIP7 chip, which can perform the matching. The binary gate has two sets of input lines (see Fig. 4.2), one for neighbourhood data and the other for control. The neighbourhood input comes from a PE's pseudo-nearest neighbours - two directly and six from special registers (edge registers), as shown in Fig. 4.8. The mask pattern is applied as the control inputs. Because of the structure of the binary gate, the pattern of '1' and '0' are compared in two separate operations. A combined result is generated in order to determine whether a pixel should be retained or deleted. If the binary gate is not utilised then to match a mask requires reading the neighbouring pixels one by one via the D-chains and comparing it with the corresponding bit in a mask. Since each Arcelli mask contains three immaterial pixels, only five neighbouring pixels are required to perform a match. Alternatively, the comparison between a mask and a pixel's neighbours can be implemented as a lookup table function.

In the Arcelli algorithm, the pattern formed by a pixel's eight nearest neighbours is compared with a set of eight masks. Each mask can be represented by a lookup table comprising 256 elements, this includes all the possible patterns can be formed from the eight nearest neighbours. The patterns are utilised as addresses to locate data within the table. To create an address from a pixel's eight nearest neighbours, the convention shown in Fig. 5.3 is applied. Due to three don't care locations in each mask, this gives rise to eight situations in which a match can be made with a pixel's nearest

0	0	X
0	1	1
X	1	X

A1

X	0	0
1	1	0
X	1	X

A2

X	1	X
1	1	0
X	0	0

A3

X	1	X
0	1	1
0	0	X

A4

0	0	0
X	1	X
1	1	X

B1

1	X	0
1	1	0
X	X	0

B2

X	1	1
X	1	X
0	0	0

B3

0	X	X
0	1	1
0	X	1

B4

Figure 5.2 The Arcelli masks

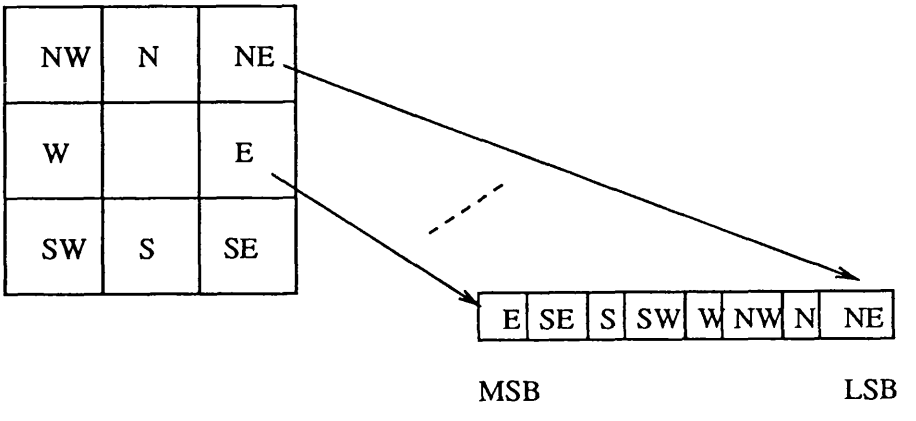


Figure 5.3 Forming an address from a pixel's 8 nearest neighbours

neighbours. Because of that, there are eight locations in each lookup table containing information that represents a match. A match is represented by a '0', whilst '1' means a mis-match. A total of eight lookup tables is required to implement the Arcelli algorithm and each table is given a unique table number (high-byte address) so that half local and half global addressing mode can be applied. The lookup tables are generated, first, by setting every location to a default value - 255. Locations in each table representing a match are calculated and '0s' are placed in those locations. After the setup of the lookup tables, the skeletonisation process takes the following steps:

1. An address is created from a pixel's eight nearest neighbours, using the convention shown in Fig. 5.3;
2. The high-byte address for Table 1 (representing mask 1) is broadcast from the controller and the address generated in (1) is used to address the Table;
3. The value is read from the location pin-pointed and is logically 'ANDed' with the pixel under consideration;
4. The result of the logical 'AND' replaces the pixel under consideration;
5. Steps from (1) to (4) are repeated for seven times but with the table number altered;
6. If the skeleton of the input image is not obtained then repeat from (1) again, otherwise the operation ceases.

In addition to the Arcelli algorithm, a fast parallel algorithm [54] employing two lookup tables to thin an image was also implemented in CLIP7A. Differing from the Arcelli algorithm, the fast parallel algorithm is devised to be implemented with two lookup tables comprising 256 entries each. The lookup tables represent two subiterations: the first aiming at deletion of south-east boundary points and the north-west corner points whilst the second removes north-west boundary points and south-east corner points. A set of rules is devised for the setup of the tables and includes:

1. Counting the number of non-zero pixels in a pixels' eight nearest neighbours; and
2. Counting the appearance of the 0-1 sequence in a pixel's eight nearest neighbours in a clockwise orientation beginning from the North direction.

As discussed above, there are two methods to perform a mask matching operation, similarly, two methods exist for the generation of an address and they concern the exploitation of the binary gate.

To generate an address through the binary gate, data is first stored in the edge registers and the Nout registers. The control inputs for the binary gate are altered so that only one neighbour pixel is examined at each time. The bits are then combined into a single address by manipulating the shifting property of the S-register.

If the binary gate is not exploited then neighbouring pixels have to be shifted in from adjacent PEs via the D-chains and examined one by one. In order to create an address, the shifting ability of the S-register is utilised. In two consecutive 3x3 windows there are six pixels in common (see Fig. 3.4). It is, therefore, possible to reduce the number of shift operations performed by manipulating the address formed in the previous window and a new convention for creating an address is adopted, as shown in Fig. 5.4. In Table 5.2 the timing results obtained from the different skeletonisation methods described above are given.

	Arcelli masks by matching without using binary gate	Arcelli masks by matching using binary gate	Arcelli masks by lookup table using binary gate	Fast parallel algorithm using binary gate	Fast parallel algorithm address generated without using binary gate
time taken per cycle (ms)	3220	333	193	49	1040
no. of masks or tables per cycle	8	8	8	2	2
process time (ms) to obtain the skeleton of Fig. 5.5(a)	45080	4662	2702	490	10400
tables set up time (ms)	0	0	20	20	20

Table 5.2 Timing results for different skeletonisation algorithms

From the results shown in Table 5.2, it is possible to deduce the time required to implement the Arcelli algorithm in lookup tables without utilising the binary gate: it will take about 58 seconds to obtain the skeleton of the image depicted in Fig. 5.5(a).

From the data presented in Table 5.2, it can be seen that the fast parallel algorithm exploiting the binary gate is most effective. This is because only two lookup tables are employed. When comparing other algorithms which also utilise the binary gate, the lookup table version of the Arcelli algorithm achieves a 42% improvement over the mask matching approach. This demonstrates the advantage of the lookup table mechanism.

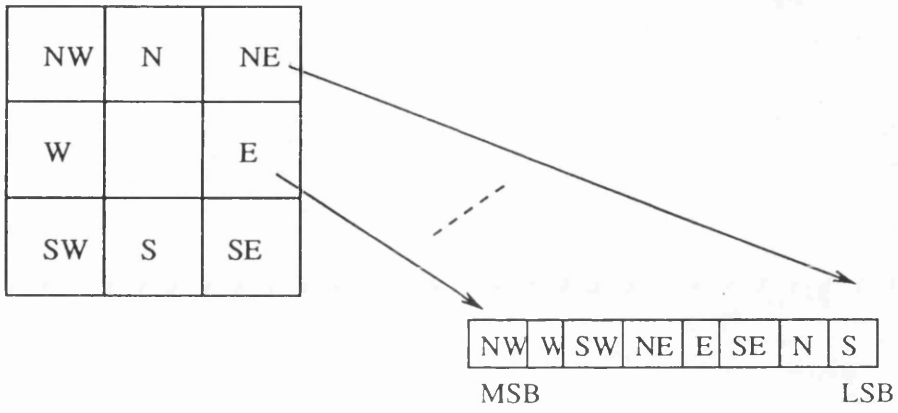
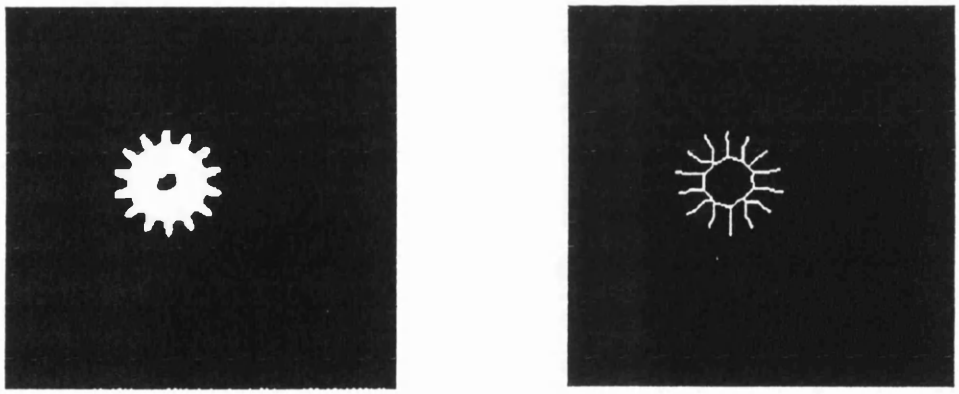


Figure 5.4 New numbering convention for forming an address from a pixel's neighbours



(a) Original image

(b) Skeleton of the gear image

Figure 5.5 Image of a gear and its skeleton

The remaining two algorithms are implemented without utilising the binary gate. The fast parallel algorithm is more efficient when comparing with the Arcelli algorithm. As described in previous paragraph, the time required to implement the Arcelli algorithm in lookup tables without utilising the binary gate will take about 58 seconds and this is longer than the mask matching version which takes about 45 seconds.

From the above discussion, it can be concluded that the fast parallel algorithm is an efficient method to perform skeletonisation and the local addressing property of the CLIP7A system enables that algorithm to be implemented. The binary gate provides a more effective way to generate addresses from a pixel's nearest neighbours and perform a mask matching.

5.2.2 Neighbourhood Reconfiguration

Neighbourhood reconfiguration is the second type of local data control and means the use of a neighbouring element as either the destination, or source, of the data. The ability to pass data conditionally between a PE and its pseudo-nearest neighbours enables the CLIP7A system to emulate different structures, e.g. a tree. A tree structure can be used for object identification, with its important application to various vision problems. In the following, an algorithm, exploiting the ability of neighbourhood reconfiguration to emulate a tree structure for simple object identification is described.

5.2.2.1 Object Identification by a Decision Tree

Object identification is among the major areas studied in machine vision, due to its wide range of applications, from quality control in factories to the identification of road objects in an automatic vehicle guidance system. Objects identifiable by such physical features as area, shape and colour, are termed simple objects but as complexity increases it is not always possible to describe them with reference to their physical features alone and they are then termed complex objects. In such cases, an object may be decomposed into segments, each segment being identified separately and the object then described from the spatial relationships between its segments. In either case, objects can be easily represented using a tree structure, as shown in Fig. 5.6 and Fig. 5.7, and the algorithm to be described is implemented for simple object identification, a brief description of how a tree structure can be applied to this follows.

A tree is a hierarchical structure beginning from the root node and terminating at the leaf nodes. Nodes in two successive layers are linked together: those in the upper layer called father-nodes, in respect of the nodes to which they are connected, and those in the lower, sub-nodes. In Fig. 5.6, the tree corresponds to a set of objects

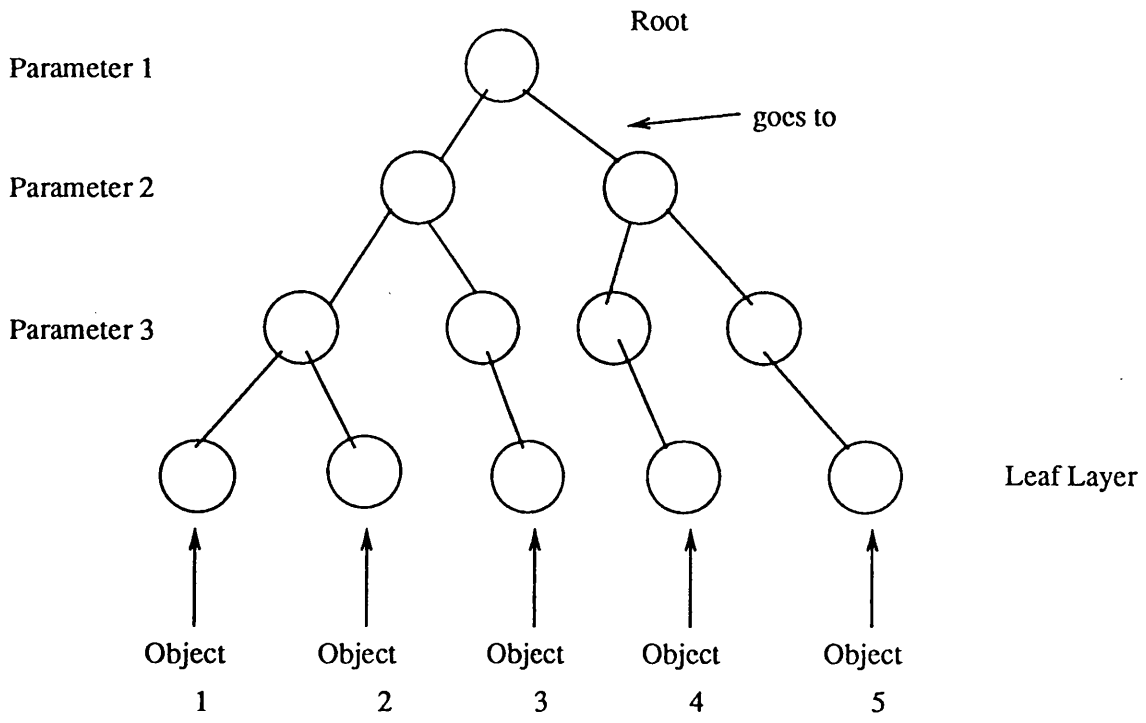


Figure 5.6 Simple objects represented by a tree

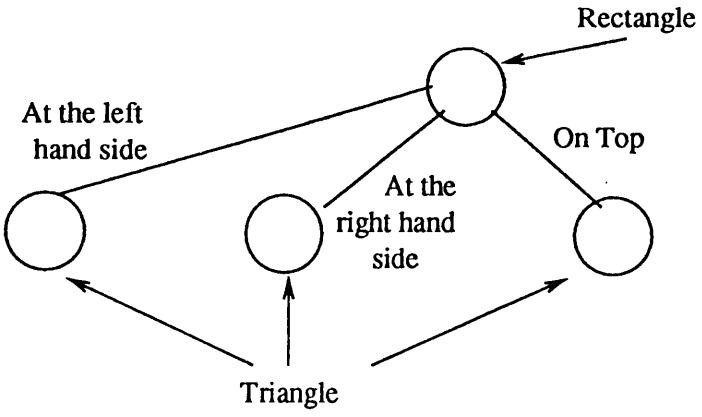


Figure 5.7 A complex object represented by a tree

represented by the leaf nodes. The number of layers employed by a tree is determined by the number of parameters used to identify the objects: each layer, excepting the leaf layer, being dedicated to a single parameter.

The method used to identify a set of simple objects is to divide the objects into groups according to their parametric values. Within a single group, objects are further classified using a second parameter and the process continues until only one object is present in the sub-group. The only condition for this representation is that no two objects have the same parametric values. Classification of objects is achieved through a comparison of their parameters with a set of discrimination factors. The discrimination factors are selected so that groups in the same category contain a similar number of objects.

Each node, except the leaf nodes, represents a group of objects: the root node, for example is the universal group containing all the objects. Starting from the root node, tests are performed in order to classify the object according to its parameters. The result of the tests indicates a node in the next layer representing the sub-group that the object belongs to. Information about parameters is then transferred to the second layer node where more testing is carried out. Only one node is active at each stage and the information flows downward until the leaf layer is reached. The active leaf node represents the input object.

does this mean its serial

5.2.2.2 Implementation of a Sort Tree in the CLIP7A System

In the above paragraphs, the way a sort tree can be employed to identify objects from their parameters, has been described. The method described was implemented in the CLIP7A system by exploiting its neighbourhood reconfigurability.

First, it is necessary to explain how the CLIP7A system can emulate a tree (a 2-D structure). When a tree is traversed from top to bottom only one layer is active at each stage. The active layer maps to a linear array. When the active node tries to link to a node in the next layer then the 3x3 neighbourhood arrangement (see Fig. 4.8) is employed. The active PE (node) loads the data into its upper edge register, which now represents the active node. The originally active PE becomes a next layer node and waits for the incoming data. Data in the edge register is then broadcast to the three PEs, representing three nodes in the next layer. Only three nodes are used because this maps well to the neighbourhood connectivity provided in the CLIP7A array. The data broadcast is stored in the Nin registers of the PEs. As described in Chapter 4 Section 4.2, each Nin register stores data received from a particular direction within the 3x3 window. The PEs are capable of selecting the data from any of the Nin registers

locally. If a PE chooses to read the data stored in the Nin register representing the North direction then it is regarded as forming a communication link with its neighbour in the North. This explains how the CLIP7A system can emulate the passing of data from one layer of the tree to the next. The object recognition algorithm is described below.

The problem is to identify an object from a set of nine simple objects. The object set is selected from the Meccano kit and the objects can be distinguished by area, perimeter and number of holes. A four-layer tree is employed and each node, except the leaf nodes, has three branches, i.e. three sub-groups. Since there are only 9 objects in the set, most of the leaf nodes are redundant.

The input image initially passes through a set of algorithms in order to evaluate its parameters. In order to simplify the problem, it is assumed that an evenly illuminated working environment is achievable and that images can be transformed into binary format by thresholding. The binary image format employed is '1' for object pixels and '0' for background and as the input images are binary, the area can be evaluated very easily by counting the number of object pixels. The perimeter is obtained by first applying an edge finding algorithm and then counting the non-zero pixels in the resultant image. Edge pixels are those having background pixels in their four nearest neighbours. To extract the edges of an object, neighbouring pixels in the North, East, South and West directions are collected by either shifting them from adjacent PEs or reading them from the local memory. By logically 'ANDing' the pixels' values, a collective result is obtained. An edge pixel is acquired by first applying an 'Exclusive OR' operation to the collective result and the origin pixel, it is followed by logically 'ANDing' the previous result with the origin pixel.

A hole in a binary image can be defined as a collection of pixels which have the same intensity with the background but are not connected to edges of the image. The holes of an object are distinguished from the background by an algorithm employing global propagation [55]. By propagating a signal '1' from the edge, the rule of propagation ensures that each cell receiving a '1' will output a '1' only if it is part of the background. This means that any background point, not receiving a signal of '1', is not connected to the edge, i.e. it is part of a hole. Evaluating the number of holes is achieved by first reducing each hole to a single pixel - the number of holes being equal to that of the non-zero pixels in the resultant image. Mask1 (Fig. 5.8) is applied to the hole image and a match with the mask causes the center pixel to be replaced by a '1'. The mask is applied to the hole image continuously until no further pixels can be added and each hole, in the resultant image, forms a right-angle in its top left hand

corner, as shown in Fig. 5.9. Mask2 (Fig. 5.10) is employed in order to retain the corner point of each hole so that each can be represented by a single pixel. The number of holes in the original image is equal to the total number of non-zero pixels in the resultant image.

The tree structure employed in this algorithm incorporates four layers and the number of nodes at each layer is 1, 3, 9 and 27, as shown in Fig. 5.11. At the leaf layer (or bottom layer), the 27 nodes are represented by 27 consecutive PEs and every three leaf nodes belongs to an unique father node in the third layer: this means that nodes in the third layer are separated by three columns. Similarly, nodes in the second layer are separated by nine columns. Since the CLIP7A system can only simulate data passing within a 3x3 window, shift operations are required to move the data to the proper PE representing a sub-node. The number of shifts required is related to the layer number.

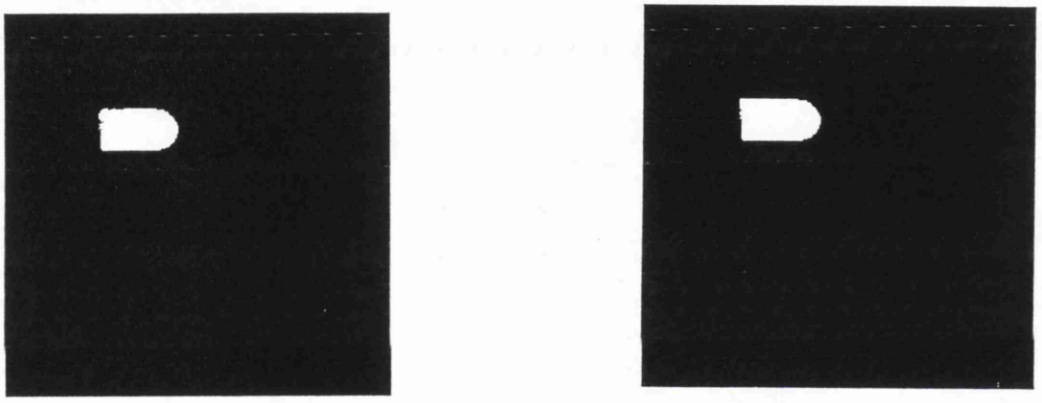
Each node in the tree is represented by a processing element in the array, and a set of three discrimination factors is stored in the memory of the corresponding PEs. The number of holes is first used to divide the objects into three groups, followed by area and, finally, perimeter. The complete sorting process is divided into the following steps:

1. The parameters of the input object are transmitted from the host to the root PE.
2. The root PE performs tests on the parametric value of the number of holes in order to select the node which the input object belongs to. Each node is given a tag representing the direction in which it is connected to its father node.
3. The root PE broadcasts the tag number to its nearest neighbours.
4. After receiving the tag number, the PEs in the next layer tune to the direction dictated by the tag number through reading the corresponding Nin register.
5. The root PE broadcasts the parametric values of area and perimeter to its sub-nodes. Because all PEs are now collecting data from a single direction but there is only one outputting data so just one PE in the next layer receives the parametric values. Data is transferred from the root PE to its sub-node in the second layer.
6. Shift operations may be required and the direction of shift is determined from the tag number, whilst the number of shifts depends on both the tag number and the layer number.
7. Steps 2, 3, 4, 5, and 6 are repeated in sequence but this time the role of the root PE is replaced by the node that received the parametric values and the area of the object is tested.

0	0	X
X		1
X	1	1

X- - - Don't care

Figure 5.8 Mask1



(a) Original image

(b) Transformed image

Figure 5.9 The effect of applying Mask1 to an image of hole

0	0	0
0	1	1
0	1	1

Figure 5.10 Mask2

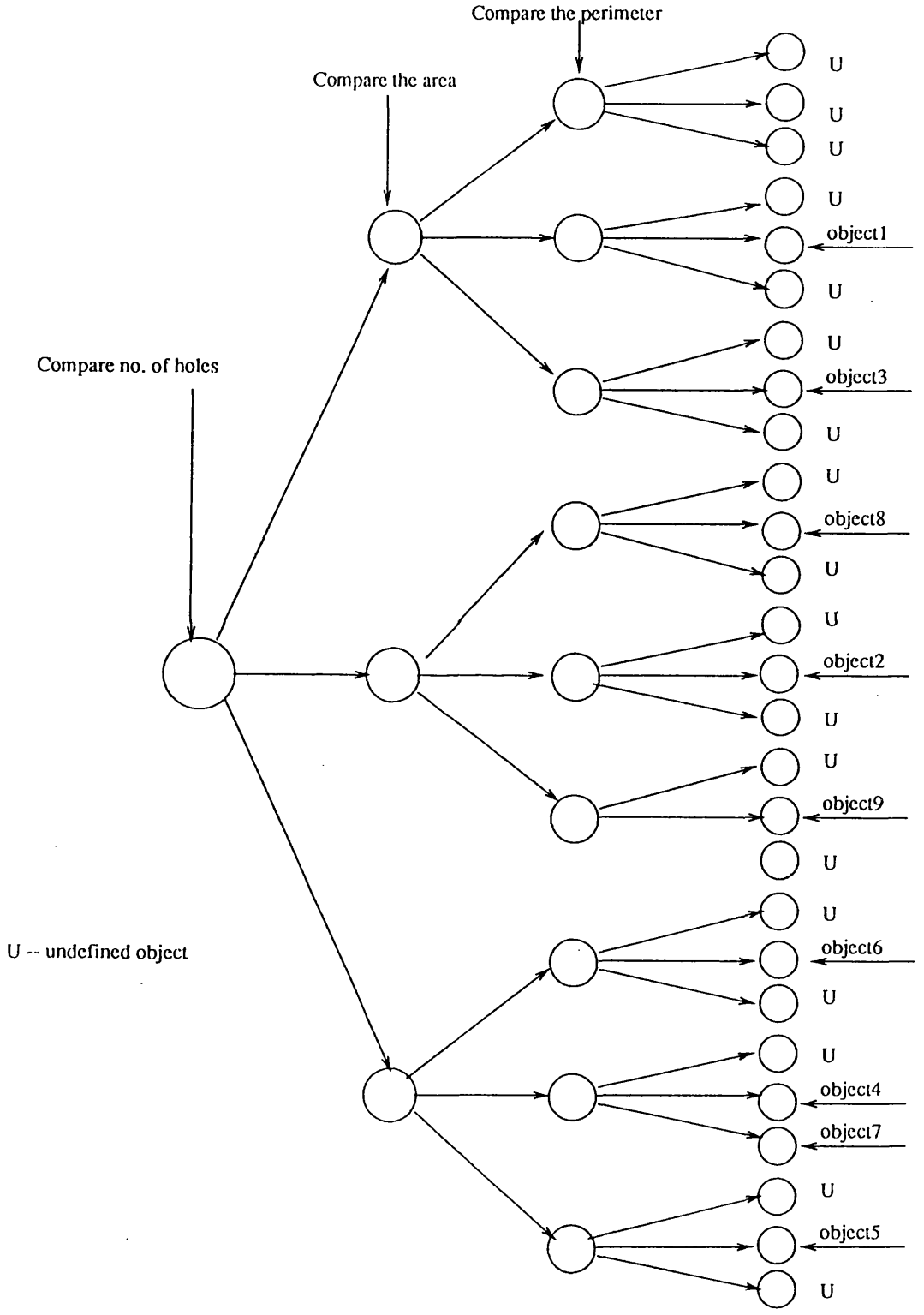


Figure 5.11 The tree structure applied to the object identification problem

8. Steps 2, 3, 4 are repeated with the node that received the value of the perimeter replacing the root PE. The tests are carried out with respect to the perimeter.
9. There are no more parameters to be transferred but the active node broadcasts a label representing 'active' to a node in the leaf layer.
10. The host examines the leaf layer and locates the position of the 'active' node via the encoding network (see Chapter 4 Section 4.3.1).
11. The position of the leaf node is used as a pointer to a lookup table containing information regarding the set of objects, thus identifying the input object.

In Table 5.3, the timing results obtained from different stages of the algorithm are depicted.

	Execution time (ms)
count area	1
count perimeter	40
count holes	6840
identify object	160
total time taken	7301

Table 5.3 Execution time for the object identification algorithm

A 4-layer tree is employed to identify 9 objects and layers can be added quite easily to accommodate more objects or when more parametric data are required to describe an object. From the execution times given in Table 5.3, the process of counting the holes can be seen to take the longest time, due to its complexity. Moreover, the execution time of the process also depends on the shape and the size of holes of an object.

The algorithm demonstrates how a linear array can emulate a 2-D structure through manipulation of the nearest neighbours arrangement. The neighbourhood connection network enables a PE to select data from within a 3x3 window. Extra devices (the edge registers) are required to store data which do not belong to the image vector being processed. Extra operations have to be performed in order to load data to these registers. Both the extra hardware and operations required are the major disadvantages of the connection network.

5.2.3 Conclusion

Two different aspects of local data control have been studied. The first was local addressing, which enables each PE to select the destination, or source, of the data from its local memory. Besides the local memory, a neighbouring element can also be used as the destination or source for the data and this is regarded as neighbourhood connectivity. The application of such an arrangement is investigated by using an object identification algorithm.

Using lookup table is one of the major applications of local addressing and two algorithms are implemented. The first explores the possibility of utilising lookup tables as a better alternative to the shift and add method of performing a multiply operation. The algorithm employs two lookup tables, corresponding to the log and anti-log tables. Multiplication involves adding the logarithms of the input data, with the result obtained via the anti-log table. The lookup table algorithm is about 18% faster than the shift and add method in multiplying two 256x256 pixel images but it has two disadvantages which concern the precision of the calculation and the setup time for the lookup tables. The setup time for the tables is 380ms and, compared with the time taken to perform the multiplication (24ms), it is a major addition to the execution time of the programme. The precision of the computation is limited to 8-bit and more local memory is required if higher precision, e.g. a 16-bit result is sought. Because of these disadvantages, the lookup table algorithm is not superior to the shift and add method.

The second lookup table algorithm is skeletonisation. Two methods are examined. The first one is the Arcelli masks method and the masks are transformed into eight lookup tables. The second method is called the fast parallel algorithm because it only uses two lookup tables to obtain the skeleton. To implement the algorithms, a pixel's eight nearest neighbours' bit pattern is used as an address to point to a location in a lookup table and removing or retaining a pixel is determined from the content at that location. It was demonstrated that the fast parallel algorithm is an efficient method because only two lookup tables are required.

Besides different algorithms, different methods that can be applied to the generation of an address from, and matching a mask with, a pixel's eight nearest neighbours' bit pattern are also examined and this relates to the function of the binary gate. From the results obtained, the binary gate improves the performance of either operation. Although the binary gate is a specialised device, its simple circuitry (see Fig. 4.2) makes it affordable to be included in future systems.

Local addressing, although it does not always improve the efficiency of a particular algorithm, makes a system more flexible and this enables the implementation of more effective algorithms, e.g. the fast skeletonisation algorithm. In later sections it will demonstrate that local addressing emulates the moving of data between PEs in the same column (as in a mesh-connected array) and this improves the performance of geometric transforms. Based on these reasons, local addressing is an important feature that should be included in the design of a linear array.

Neighbourhood reconfiguration enables a PE to select data coming from a 3x3 window. In a linear array, data from the eight directions in a 3x3 window come from three consecutive rows of image data. In order to emulate the passing of data from all eight directions extra registers are needed to store data from the other rows. Besides the extra hardware, extra operations are also required to load the data to the data registers. Since each PE in the CLIP7A system processes data in the same image column, so data coming from the eight directions can be obtained from its left or right neighbours and this can be achieved by a simple shift operation. Neighbourhood reconfigurability is not an effective mechanism for a linear array because the nearest neighbours connectivity is very different from that of a 2-D array. Moreover the extra hardware required also makes it undesirable. In any future design, improving the data transfer rate between neighbouring elements will be more important than emulating the eight nearest neighbourhood connectivity.

5.3 Structural Properties of a Linear Array

Image processing in CLIP7A is performed in a scanning format and one operation must be repeated 256 times in order to process the entire image. An image can be roughly divided into areas of interest and areas of no interest. If an image operation only concerns areas of interest then the number of iterations required to carry out such an operation can be reduced by scanning over those areas only. A pre-processing algorithm which can locate all areas of interest in a binary image was implemented. The effectiveness of the algorithm is examined by comparing the performance of a skeletonisation operation applied to the entire image and only to the objects appearing in an image.

Additionally, data passing between elements in a linear array is investigated. Data transfer is an important factor in the design of a multi-processor system. The CLIP7A system provides two special features for data transfer: local addressing and dual communication channels which require further examination. The effectiveness of these features is studied via two geometric transforms.

5.3.1 Pre-processing Algorithm

The goal that the pre-processing algorithm tries to achieve is to determine the minimum number of iterations that is required to perform an image operation. It is first required to identify the areas of interest and then determine the number of repetitions for the operation by counting the total number of rows occupied by those areas. In order to simplify the problem, only binary images are considered and objects are represented by pixels of '1', whilst background has pixels of '0'. If grey-level images are considered then a different algorithm is required. The effectiveness of the pre-processing algorithm also depends on other factors which are independent of the types of images under consideration and further discussion is included in later paragraphs. Using a binary image as an example provides sufficient information to determine the effectiveness of such an algorithm.

The most straightforward method to evaluate the number of iterations required is by counting the number of non-zero rows of the input image, because the background has value 0. However, such a method will not cater for more complex situations and three different cases should be considered.

The three different cases include images which have a single object, multiple objects without overlap, and multiple objects with overlapping, as shown in Fig. 5.12. The term overlap applies to any objects which have pixels located in the same column (or the X coordinate). If there is only one object appearing in the input image then the number of iterations is equivalent to its height (or the number of rows occupied by it). When more than one object is present, without overlap, then the height of the tallest object will determine the iterations needed. Overlapped objects are treated as a single object, as shown in Fig. 5.12(c). This is because pixels in the same column are processed by a single PE, therefore, there is no easy solution to operate on different objects having pixels in the same column in parallel. In the last case, the number of iterations is still determined by the height of the tallest object but now overlapping objects are treated as a single entity.

The algorithm first scans through the image from top to bottom and a vector V1 which stores the Y coordinate of the first pixel of each object is obtained. This is followed by a bottom up scan and this time vector V2 is generated. V2 contains all the Y coordinate of the last pixel of each object. The maximum height of the objects will determine the number of repetitions of a single operation that must be applied. The maximum height is evaluated by subtracting V1 from V2 and the maximum value in the resultant vector is equivalent to the maximum height. Objects are shifted to the top of the image and operations are applied to the objects with the number of repetitions

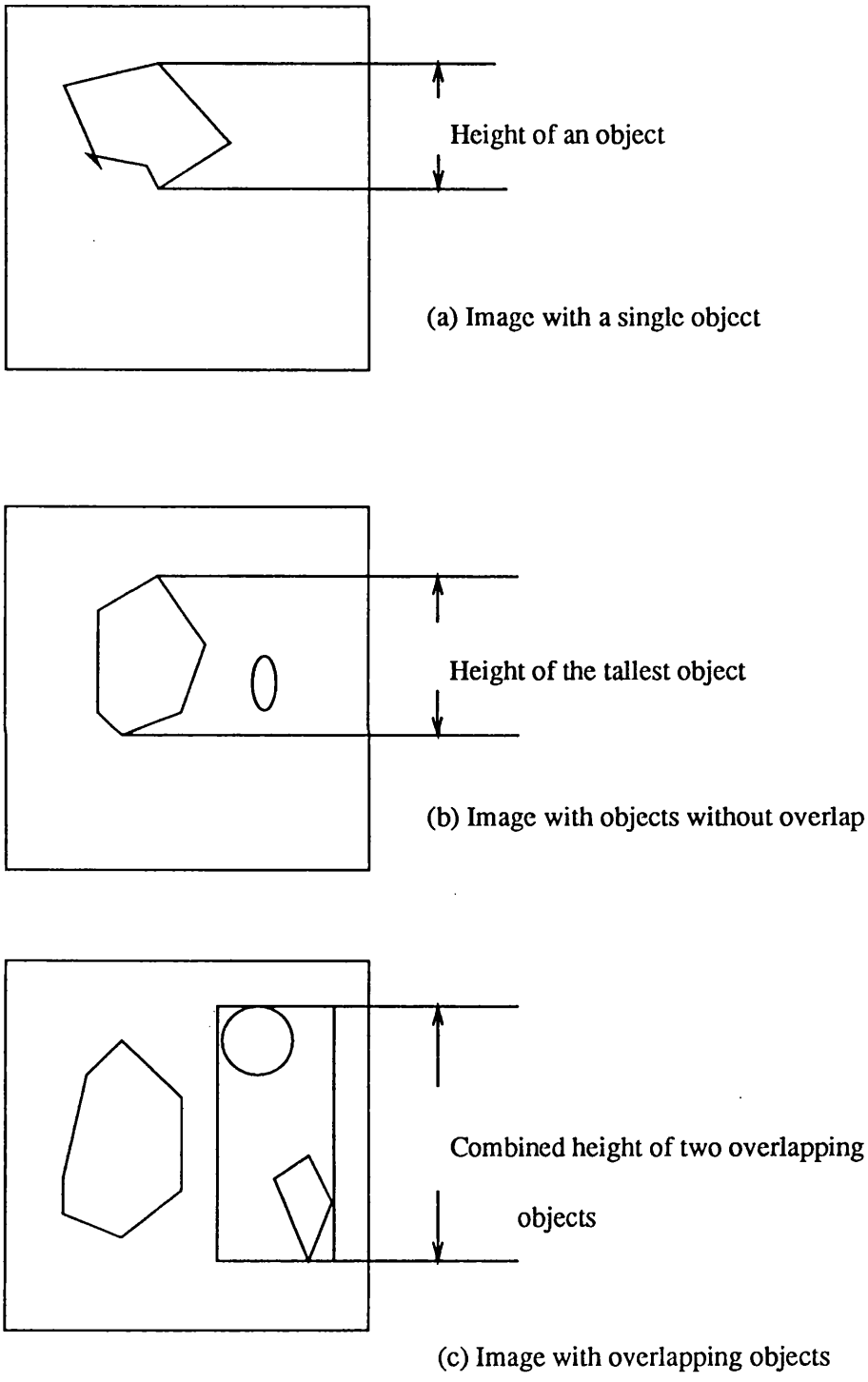


Figure 5.12 Images considered in the pre-processing algorithm

determined above. Fig. 5.13 shows how pre-processing can be applied to a binary image containing two objects without overlap and Table 5.4 summarises the timing results obtained from performing the skeletonisation of the image shown in Fig. 5.13 with and without applying the pre-processing function.

	thinning with pre-processing algorithm by fast parallel algorithm	thinning without pre-processing by fast parallel algorithm
no. of repetitions	82	256
time to thin image (ms)	160	500
no. of tables	2	2
pre-processing (ms)	180	0

Table 5.4 Execution time for thinning with and without pre-processing

The execution time of the pre-processing algorithm takes 180ms and the thinning operation takes 160ms so altogether there is an 32% improvement over the original method. There are conditions in which the pre-processing algorithm is not practical. The system scans along the Y-direction, so if an image is densely occupied by object(s) along the Y-direction then the number of repetitions is still large. If a simple operation is to be carried out in the image, e.g. edge finding, then the execution time of the pre-processing algorithm will outweigh the time saved by the reduction of the operation. The effect of applying the pre-processing operation can be expressed by Equation (5.3).

$$T_t=T_p+T_f\frac{(x)}{256} \tag{5.3}$$

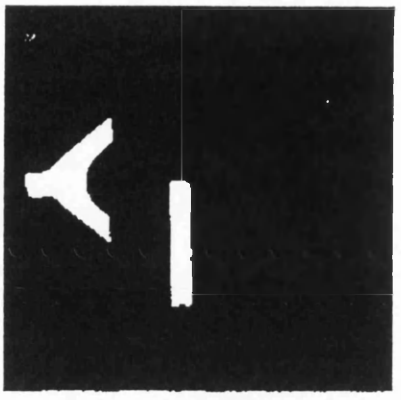
T_t = total time taken for the complete operation

T_P = time taken for the pre-processing operation

T_f = time taken for the image operation(s)

x = number of repetitions

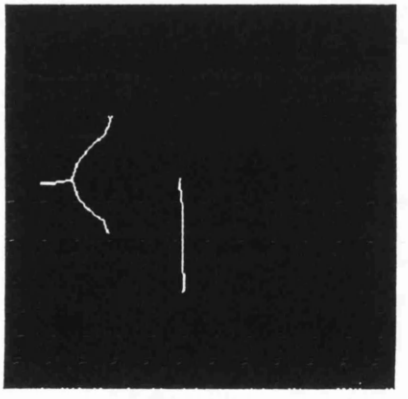
When pre-processing operation is not applied then $T_t=T_f$. The term T_p can be regarded as a constant, but varies with different image formats. If T_p is relatively



(a) Input image

(b) Objects shifted to the top of an image

(c) Skeletons of the objects obtained



(d) Skeletons moved to the original positions of the objects

Figure 5.13 Operation sequence of the pre-processing algorithm

small then its effect on the term T_t is negligible and T_t can be considered as only proportional to both T_f and x. Dividing (5.3) by T_f, equation (5.4) is obtained.

$$\frac{T_t}{T_f}=\frac{T_p}{T_f}+\frac{x}{256} \tag{5.4}$$

The term $\frac{T_t}{T_f}$ is the ratio between the total time taken to perform an operation with pre-processing and without. If $\frac{T_t}{T_f}$ is small then pre-processing is effective. As discussed above, this depends on the value of T_f and x, i.e. the execution time of an image operation and the number of iterations required, after pre-processing, to complete such an operation when the value of T_p is small. When T_f is large and both T_p and x are small then pre-processing is advantageous.

5.3.2 Image Transforms

Image transforms are interesting operations with which to examine the performance of an image processor because of the amount of data transferred between processing elements. In the CLIP7A system, pixels in the same image column are stored in the local memory of a single PE and accessing data within a column takes a constant time which is equal to the memory access time. Additionally, there are two 8-bit communication channels in the system and such an arrangement can improve data transfer along a row of image. In a mesh-connected array, accessing data residing in an element in the same column, or row, requires the shifting of data from one PE to another and the number of shifts required is equal to the distance between the two elements. In the following, two image transforms will be discussed: reflection and rotation.

Reflection

Two types of reflection are considered: with respect to a horizontal line and a vertical line. It is possible to represent the two reflections mathematically by

$$ReH(x,y)=Im(x,(255-y)) \tag{5.5}$$

$$ReV(x,y)=Im((255-x),y) \tag{5.6}$$

where ReH and ReV represent resultant images of reflection with respect to a horizontal line and a vertical line respectively, and Im is the input image.

In the case of reflection with respect to a horizontal line, pixels are only transferred in the Y-direction and this is equivalent to moving pixels of data from one

memory location to another. For the CLIP7A system, this data manipulation can be achieved by repeating a single operation: vector move. By repeating the move operation 256 times, the reflection with respect to a horizontal line of the input image (Fig. 5.14(a)) is obtained and shown in Fig. 5.14(b).

The reflection of an image about a vertical line involves the movement of data in the X-direction and, in the CLIP7A system, this is accomplished by moving data along the D-chains. As described in Chapter 4 Section 4.3.1, the two D-chains can be connected in two different manners: *wrap-around* and *reflect*. The *reflect* connection is used, and by shifting the data 256 times in both D-chains the reflected data of a vector loaded to the co-processor's D-chain is obtained in the processor's D-chain. Similarly, the reflected data of the vector originally residing in the processor's D-chain is now available in the co-processor's D-chain. By employing both D-chains, the number of operations required to process the entire image is halved, and Fig. 5.14(c) depicts the reflection of the input image with respect to a vertical line.

Rotation

Image rotation involves the moving of pixels in both X and Y directions. The relationships between the resultant image and the input image can be expressed by the following equations:

$$X' = X\cos A - Y\sin A \tag{5.7}$$

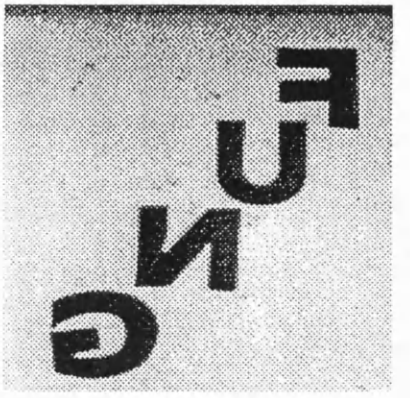
$$Y' = X\sin A + Y\cos A \tag{5.8}$$

where X', Y' are the pixel co-ordinates after the transform, and A is the angle of rotation. As shown in Fig. 5.15, a rotation can be performed in two steps. The first step is moving the pixel from (X,Y) to (X,Y'); the second step will move the pixel from position (X,Y') to (X',Y'). Pixels with a vertical displacement outside the image boundary will be lost after the rotation, as shown in Fig. 5.16. This method need not be applied to a rotation of 90 degrees or its multiples as discussed in [56]. When the angle of rotation is 90 degrees then from equation (5.8), Y' = X. The first step to move (X,Y) to (X,Y') becomes moving (X,Y) to (X,X). All the pixels will be moved to the diagonal of the image. Rotation of 90 degrees or its multiples has to be handled separately. In the following, an algorithm which performs rotation of 90 degrees is described.

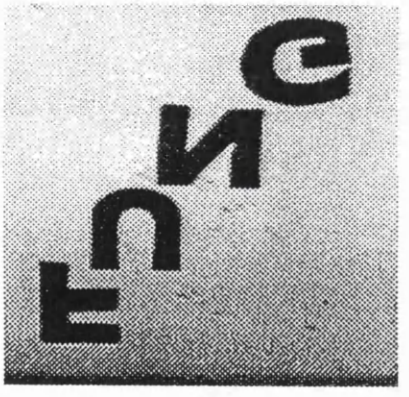
In [57], an algorithm is given which breaks down the rotation into three shearing operations which can be expressed by the following equations:

$$Sh1(x,y) = Im((x+y),y) \tag{5.9}$$

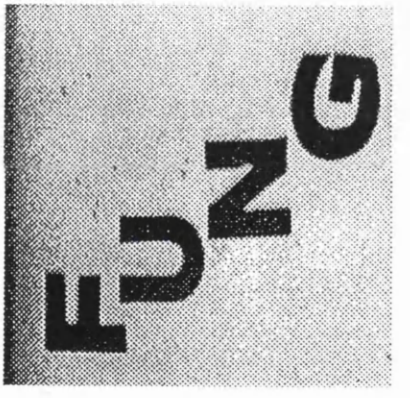
(a) Input image



(c) Reflection with respect to a vertical line



(b) Reflection with respect to a horizontal line



(d) Rotation by 90 degrees

Figure 5.14 Resultant images from different geometric transforms

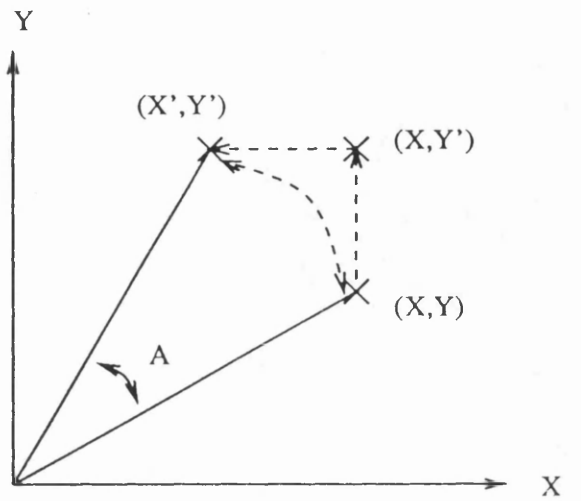
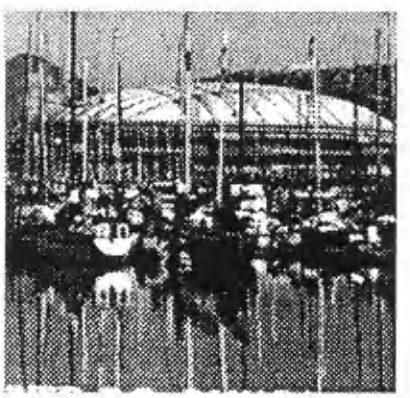
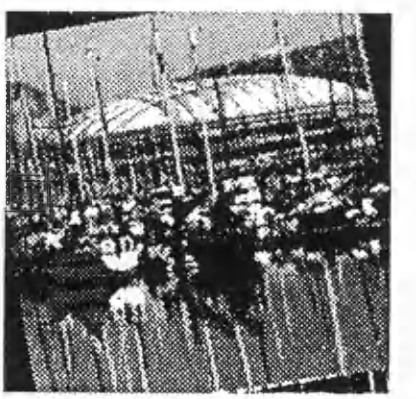


Figure 5.15 Performing a rotation by two move operations



(a) Input image



(b) Rotated image

Figure 5.16 Rotation by 10 degrees

how long does this take?

$$Sh2(x,y)=Sh1(x,(x+y+1)) \tag{5.10}$$

$$Sh3(x,y)=Sh2((255-y+x),y) \tag{5.11}$$

The top left corner pixel has the coordinate (0,0). The X coordinate increases towards the right hand side, while the Y coordinate increases downward. The values for both X and Y are modulated by 256. The equations show that at each shearing operation only transformation in a single direction is necessary and this can be easily implemented in the CLIP7A system. The first shearing operation is to perform right shift, with the position shifted by each row equal to the row number of an image vector. The second operation involves the moving of data between different memory locations within the same processing element. The final shearing is a left shift and the step for each shift is related to the row number. The first and third operations are performed by manipulating the D-chains, in *wrap-around* mode. The second shearing can be achieved by local addressing. The resultant image after the rotation of 90 degrees of the image depicted in Fig. 5.14(a) is shown in Fig. 5.14(d). Table 5.5, summarises the timing results for the image transforms discussed.

Operation	CLIP7A (ms)	Pseudo CLIP7 mesh (ms)
Reflection with respect to a vertical line	40	43
Reflection with respect to a horizontal line	20	80
90 degrees rotation	40	130

Table 5.5 Timing results for image transforms

The timing results of the image transforms obtained from the CLIP7A system are compared with the emulated results of a pseudo 2-D mesh-connected CLIP7 array. The pseudo 2-D CLIP7 array embodies 65536 PEs arranged in a 256x256 mesh and each PE consists of a single CLIP7 chip. Horizontal data transfer is via a single D-chain whilst vertical data transfer is through the neighbourhood registers. The control of the pseudo system is SIMD but an element can be inhibited by a value in a local register.

From the results presented in Table 5.5, it demonstrates that the geometric transforms can be performed more efficiently in the linear CLIP7A array than the mesh-connected array. The advantage of the linear array in performing reflection with respect to a horizontal line is prominent. This is because the moving of data in the same image column can be achieved by one read and one write operation of the local memory. For the mesh structure there are two factors that affect its performance. The first one is the vertical data communication between elements. The neighbourhood registers, Nin and Nout, are less efficient because they employ bit-serial data transfer. The second factor is related to the SIMD nature of the mesh structure. In performing the reflection function, the displacement of each datum varies with respect to its original position and after each shift operation only two pieces of data should be stored in each column. Because of the SIMD controlling mechanism, during the write operation only two PEs in each column are active. Extra operations are required in order to select the proper PEs. In the above exercises, masks are utilised to enable, or disable, a PE. Masks are generated and stored in the local memory of the PEs. Each mask can enable a selected pair of PEs in a column and by referring to the proper mask after each shift operation the reflection of an image is obtained. Such an arrangement requires extra access of the local memory and this degrades the efficiency of the system.

In performing reflection with respect to a vertical line, because of the linear array's dual D-chains and switchable data path, the linear array is more efficient, taking into account the number of PEs provided. The major drawback of the mesh structure is also the requirement to disable and enable the PEs after each shift operation because the displacement of each datum is also dependent on its original position (refer to equation (5.6)). The horizontal data communication of the mesh structure is through a single D-chain which is a 8-bit channel and this is the reason for a more efficient performance in obtaining the reflection with respect to a vertical line than to a horizontal line.

The rotation function involves data transfer both horizontally and vertically. In both horizontal operations, pixels in the same row make the same number of displacements depending of their row number (see equation (5.9) and (5.11)). In the first shear, for example, pixels in row 0 do not shift, whilst each pixel in row 1 shifts one pixel to the right. In the second horizontal shear, pixels in row 0 move one step to the left and two shifts are performed by row 1. For the vertical shearing operation, pixels in the same column make the same number of moves according to their column number (see equation (5.10)). As an example, pixels in column 0 move one step downward and in

column 1, two shifts are required. Because of the different number of horizontal, or vertical, shifts to be performed, as described previously, masking is required to enable the proper PEs after each shift operation and this degrades the performance of the pseudo-system. For the linear array, both D-chains are utilised during the horizontal shearing operations. In the vertical shearing operation, local addressing is exploited.

It is, however, necessary to point out that in operations which involve moving data horizontally for the same distance then the mesh structure will be more effective. To shift an image ten pixels to the right, for example, the mesh structure will be 256 times faster than the linear array.

5.3.3 Conclusion

The structural properties of a linear array have been examined in two separate areas. In the first case, the scanning property of a linear array is contemplated. In a linear array, an operation has to be repeated 256 times in order to process the entire image. When part of an image does not carry useful information then the number of repetitions can be reduced by ignoring those areas. A pre-processing algorithm can be implemented in order to locate areas of interest. Once the areas of interest are determined, then image operations will be applied to those locations in an image. The effectiveness of the pre-processing algorithm is governed by three factors: execution time of the pre-processing algorithm, execution time of image operations and reduction in scan. If the first factor is low and the other factors are high then the pre-processing algorithm will improve the performance of the system significantly. The execution time of the pre-processing can be regarded as a constant which may be governed by the image type being processed. The execution time of the image operations and the reduction in scan may vary with respect to the problem. Whether the pre-processing algorithm should be applied or not should be determined after a study of the nature of the problem.

The second case examines the moving of data between PEs in a linear array. Three geometric transforms are implemented and they involve moving data in either the same row or in the same column. The timing results obtained from the CLIP7A are compared with the emulated results of a pseudo 2-D array which embodies 65536 CLIP7 chips arranged in a 256x256 mesh. The results indicate that the linear array is more efficient in performing those geometric transforms, because of:

1. The dual D-chain structure and its switchable connectivity; and

2. The ease of accessing local memory either by local addressing or global addressing.

It is the special hardware features that give a linear array its advantages, and they will be considered in the design of a future linear array.

5.4 Summary

In this Chapter, the structural properties of linear arrays and local autonomy have been studied. Although the examination is not exhaustive, useful information which concerns the design of linear arrays has been obtained. The conclusions derived from local autonomy and the structural properties support each other. Local addressing and the dual communication channel configuration will be included in the design of a future system. Local addressing improves the flexibility of a linear system so that more effective algorithms can be implemented. It also allows the emulation of passing data between PEs in the same column as demonstrated in the image transforms. The dual communication channel structure will improve the efficiency in moving data between neighbouring elements and this improves the performance of the geometric transforms. Additionally, the function provided by the binary gate will be embodied.

The next chapter describes the investigation of the processing requirements imposed by intermediate-level and high-level operations, which have not yet been considered. Further, an attempt will be made to establish a structure which maps well to the overall vision problem.

Chapter 6
Image Processing in the CLIP7A System II

6.1 Introduction

The algorithms described in Chapter 5 are devised to study the structural properties of a linear array and the effectiveness of local autonomy. The information obtained will assist the design of the linear arrays included in the multi-layer system. The next step towards the development of a multi-layer system is to examine the relationship between the nature of the problems and the overall structure of the system and this includes two major areas:

1. The processing requirements at different levels of operations; and
2. Relationships between the structure of a problem and the architecture of a system.

The approach adopted in this investigation is to apply the CLIP7A system to solve an image processing problem. As the multi-layer system is devised to tackle vision problems comprising operations in low, intermediate and high level processing, four such problems are considered and a brief description of each follows.

The first example is 2-D electrophoresis gel analysis. Electrophoresis gels are devices used in the analysis of protein mixtures, often in connection with cancer research. An image of an electrophoresis gel may contain up to several hundred grey spots which represent the presence of various protein molecules in a sample. The position of a protein molecule in the gel is governed by its acidity and molecular weight. The acidity affects the position of a molecule in respect of the X axis, whilst the molecular weight determines the location of a molecule along the Y axis. An analysis of the gel includes quantification of the amount of protein present and the comparison of the gel with a standard sample. A survey of various methods applicable to the analysis of the gel can be found in [58].

The quantification process will generally involve the detection of the gel spot at which point filtering and segmentation are the two major operations to be performed. The coordinates, area, integrated optical density are among the properties that must be extracted from the gel image. After the quantification process, each protein spot is represented by these data. The identification of protein molecules appearing in the input gel can be achieved by mapping the input to a sample gel. Mapping one gel to another is extremely complex due to the variable location of a protein molecule caused by its sensitivity to minor changes occurring during the production process.

Comparison of two gels will usually involve the manual identification of some spots before automatic mapping can take place. The manually located molecules can then be used as markers which are necessary in order to align the gels. The sample gel is used as a reference and a geometrical transform is applied to the input gel so that the two can be aligned. The transformation required is determined by the differing coordinates between manually located molecules. Once the gels are aligned, comparison between the protein molecules can proceed.

The second example is human face profile recognition. This has a practical importance in many applications, such as security checks. The face profile can be obtained quite easily by correct control of the light source, as described in [59]. The recognition process can be divided into three different stages. In the first stage, interesting points (called fiducial points), important for the characterisation of outline curves, are located. The number of fiducial points extracted is different in different algorithms; 6 and 11 are used in [59] and [60] respectively. The nose peak is a typical example of a fiducial point and can be located by searching for the right-most point in the outline curve (assuming the profile is facing the right hand side). Some fiducial points, such as the nose bottom and the mouth point, can be located by a corner detection algorithm as in [59], whilst others are located using some pre-determined criteria. As an example, the distance between two prominent points such as the nose and the chin point, can be used as one criterion to locate other fiducial points.

The second stage involves the evaluation of a set of parameters based on the coordinates of the located fiducial points. The definitions for the parameters are different in various recognition methods and some examples are:

1. The distance between two neighbouring fiducial points,
2. The angle defined by three consecutive points.

A database file is created to accommodate all the parametric values evaluated from the group of profiles to be identified by the system. Values stored in the database file should be able to cater for minor discrepancies, such as slight movement of the object during the input of the profile. In order to achieve this, several profiles of each object are taken during the construction of the database and the parameters for each profile are computed. The mean and standard deviations for individual parameters are evaluated and stored in the database which is then used to determine the best matched profile to the input profile. This is the third, and final state of the recognition process and includes computing the differences between the parameters in the database and the input profile. Each profile embodies a set of parameters and a collective result is required in order to indicate the likelihood of the input profile being present in the

database. The differences between individual parameters from the input profile and the database values are added together and the accumulated result is used to determine the best matched profile.

The next example is Chinese character recognition, in which field research has been carried out for more than 25 years. In [61], a brief history of the topic together with a survey of different recognition approaches currently being studied is given. The recognition process will depend on the encoding method employed and only the stroke matching approach is described in here.

The stroke matching approach uses strokes of a character as the features to describe the character and different recognition algorithms can be found in [2] and [62]. Generally speaking, the process can be divided into three stages: pre-processing, feature extraction and matching. The methods described in [2] and [62] both use the skeleton of the input character to represent the character itself and thinning is part of the pre-processing algorithm. Hough transform or template matching can be utilised to extract the stroke from the skeleton and each stroke will be described by a set of parameters including its length, orientation, etc. At this stage the input character will be described by the parameters of the detected strokes. The matching process involves searching through the database in order to identify the input character by comparing the parameters of each stroke. An overall match strength can then be evaluated. Characters to be recognised belong to a huge set which can embody up to 56000 characters. This means that the database file carrying all the parameters of the character set is enormous and an effective searching algorithm is consequently vital.

The last example is the second DARPA benchmark problem [63]. The DARPA benchmark is designed to evaluate the merits of various parallel architectures applied to image understanding. The benchmark problem is tailored to embody processing at all three levels and algorithms to be performed at each level are well defined. The problem requires the identification of an input scene with respect to a set of models. The input scene includes a set of two images providing intensity, and depth, information respectively. Rectangles are the only objects that exist in both images and the input scene is defined by the spatial relationships between the rectangles. The operation sequence can be roughly described as identification of rectangles followed by model matching. A rectangle is identified by its corners and a good candidate rectangle should have four right-angle corners. Corners of rectangles appearing in the intensity image are located by the k-curvature method [64], and then verified by the angles formed by the convex hull of the rectangles. Corners in the depth image are evaluated

according to the intersections of line segments extracted after a Hough transform. After the extraction of right-angle corners, a rectangle is described by parameters, computed from the coordinates of the corners. The parameters include the length of the axes, the orientation of the rectangle and the coordinates of the centre. Model matching is a two step process. First it includes the matching of the model rectangles to the candidate rectangles according to their parameters. Only those candidate rectangles that show a positive result when matched with the model rectangles will be utilised in the second matching process. This includes the matching of the spatial relationships between candidate rectangles and those of the model rectangles. The input scene is identified by counting the number of the spatial relationships being matched to the model. A detailed description of this problem can be found in Section 6.2.

Despite the diversity of the above examples, they all share the same problem structure and embody processing at every level. The data types evolved at each level can be roughly divided as image, symbol, and object. Excepting images, there is no clear definition for these: what a symbol represents depends on the nature of a particular problem and this also applies to an object. For example a symbol can be a corner or a fiducial point, whilst an object can refer to a character or to a face profile. The relationships between the level of processing and the data type involved is illustrated in Fig. 6.1. A general structure for an integrated problem is thus suggested, but it remains uncertain how a linear array can be applied to operations at each level or how elements at different layers communicate in order to map the physical structure to the problem. To derive answers to those queries, the CLIP7A system is once again applied to solve one of the problems. Because of the similarity in structure of the problems, one of them is chosen as representative. The DARPA benchmark problem is selected as the sample problem because the structure of the problem is well-defined, so that any architectural issues related to the structure of the problem can be easily observed.

In the next sections, algorithms developed to tackle the benchmark problem will be described. It is important to emphasise that the object of the exercise is not the benchmarking of the CLIP7A system. Attention should be focused on the architectural requirements imposed by such a scene analysis problem.

6.2 The Second DARPA Benchmark

The second DARPA benchmark problem is designed as an integrated image processing problem to include all three levels of image operations. The following sections will summarise the recommended method described in [63] for solving the

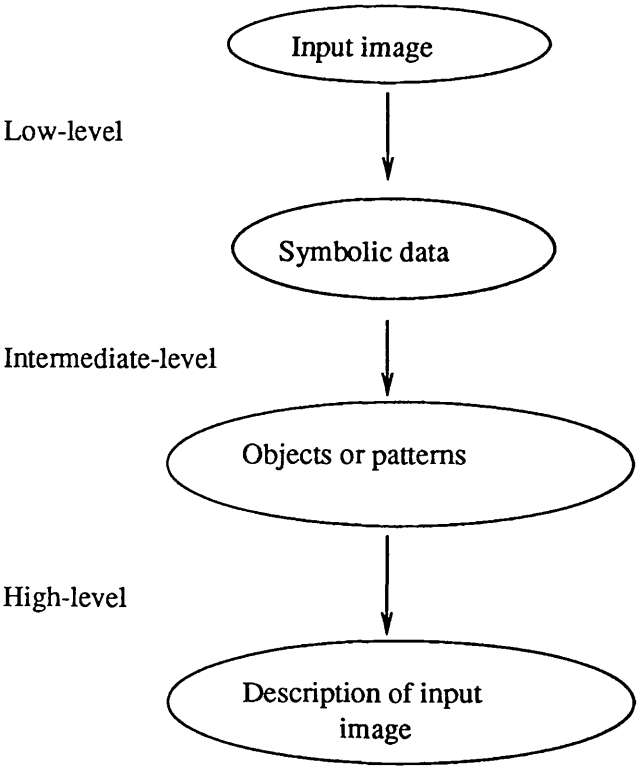


Figure 6.1 A generalised structure of an integrated vision problem

problem. This will highlight the complexity of the problem.

The benchmark can be regarded as a scene analysis problem and the aim of the exercise is to construct a description of the input scene and then match it to the models provided in a database. Information regarding the intensity and depth of the input scene is conveyed in two different images. The intensity image is a 512x512 array of 8-bit integer values, whilst the depth image is a 512x512 array of 32-bit floating-point values. The intensity image is noise free, while the depth image has added Gaussian noise. As shown in Fig. 6.2, the image displays a collection of rectangles and the spatial relationships between these are utilised in the description of the input scene. Information concerning a model is split into two sections - the parameter table, and the relational table. The parameter table provides information concerning the physical features of the model rectangles appearing in the model. The relational table consists of data governing the spatial relationships between the model rectangles. The terms used in both tables can be explained graphically as shown in Fig. 6.3. An example of a model file is presented in Table 6.1.

Because spurious rectangles are deliberately added to both images, the matching process is more difficult. As mentioned above, the solution for the problem includes processes to be performed at all three levels. A brief summary of those operations to be executed at each level follows.

6.2.1 Low-level Processing

For the intensity image, low-level processing includes both labelling and a corner detection algorithm. Each connected component in the intensity image is given a unique integer label. The K-curvature method is used to locate the right-angle corners from the boundary of the labelled components.

The depth image is initially median filtered to remove noise and the smoothed image is stored, to be referred to at a later stage. Strong edges of the rectangles in the smoothed image are obtained by two operations: Sobel operator and thresholding. Two images: the smoothed depth image and the edge image are generated after the low-level operations.

6.2.2 Intermediate-level Processing

The goal of intermediate-level processing is to extract good candidate rectangles from the output of the low-level processed intensity image. The rectangle hypothesis is based on the number of right-angle corners obtained from the K-curvature values. If a connected object has three contiguous right-angle corners then it will be regarded as

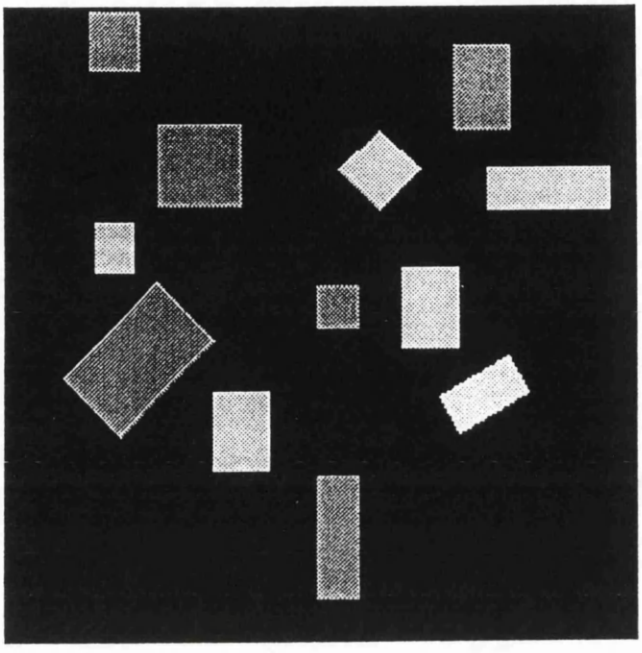


Figure 6.2 An intensity image of the benchmark problem

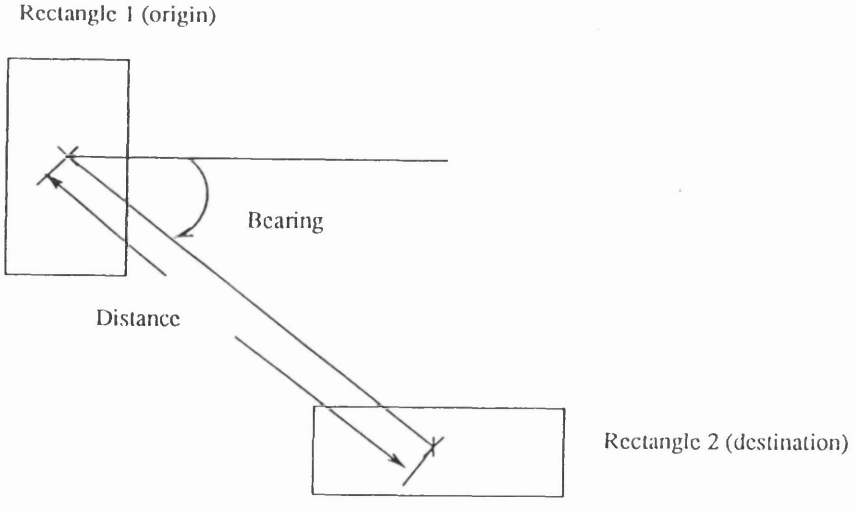


Figure 6.3 Spatial relationships between two rectangles

major axis	minor axis	orientation	intensity	depth
48.0	16.0	0.0	163	0.0
20.0	16.0	30.0	163	57.0
33.0	26.0	45.0	103	100.0
40.0	19.0	0.0	120	200.0
50.0	45.0	58.0	233	0.0

The parameter table for the model rectangles

origin	destination	bearing	distance
0	1	200.0	163
0	2	310.0	16
2	3	45.0	63
3	4	120.0	50

The relational table for the model rectangles.

Table 6.1 An example of a model data file

a candidate rectangle. The convex hull of the candidate rectangle is obtained and the hull angles are used to verify the right-angle corners derived from the K-curvature values. If there are again three right-angle corners existing in the convex hull then the candidate rectangle is confirmed, if not it is rejected. The parameters of a candidate rectangle are then evaluated from the X and Y coordinates of the corners and the definitions of these can be found listed in Table 6.2.

Hough transform is used to extract axes of rectangles from the edge image, which is derived from the depth image subsequent to low-level processing. If all the axes of a rectangle are located from the edge image then the presence of a rectangle is confirmed and parameters of the rectangle are evaluated. This process is directed by high-level processing and Hough transform is only applied to a sub-area of the edge image.

major-axis	length of the longer side
minor-axis	length of the shorter side
orientation	angle of the major axis with respect to the X-axis
centre	pixel coordinates nearest the midpoint of the diagonal

Table 6.2 Definitions for the rectangle parameters

6.2.3 High-level Processing

The first goal of high-level processing is to match, in terms of size and intensity, the candidate rectangles, extracted from the intensity image only, to the model rectangles. The spatial relationships between the matched candidate rectangles are evaluated and compared with a model's relational table. After the matching process, the depth information is used to confirm the matched candidate rectangles one by one. A sub-area in the smoothed depth image (resulted from the low-level operation) where a candidate rectangle should be found is located and the pixels' values within the area are counted. If the number of pixels (within the area) which are too deep exceeds a threshold value (determined from the model data) then the candidate rectangle is rejected and any matches it made with other candidate rectangles will be rendered void. If the candidate rectangle is not rejected then a match strength is evaluated. The match strength is the percentage of pixels both in the correct depth range and with the correct intensity, with the percentage of those that are too deep subtracted.

At this stage all candidate rectangles are extracted from the intensity image and verified with the depth information. Because of the presence of spurious rectangles, it is unlikely that the input images can be identified at this stage and an examination of the depth image by a top-down approach is carried out. In order to continue the matching process, other candidate rectangles are located in the depth image. A window (with its position and size derived according to the model data) where a missing rectangle is likely to exist in the depth image is then considered. Hough transform is applied to the window in order to extract the axes of the rectangle, as described in Section 6.2.2. If a rectangle is found, its parameters are evaluated and the matching process continues. A match strength is evaluated for the newly detected rectangle. The probing of the depth image continues until no more rectangles can be found. The best matched model will have the highest average match strength obtained from the candidate rectangles which match with its model rectangle.

6.3 The DARPA Benchmark Problem as a Scene Analysis Problem

As described in Section 6.1, the DARPA benchmark is used only as an example of a scene analysis problem and the objective in trying to solve the problem is not the benchmarking of the CLIP7A system. With this in mind the following modifications have been introduced:

1. The size of both the intensity and depth image is reduced to 256x256 pixels.
2. Instead of a floating-point representation, 8-bit integer is used to represent the depth data.
3. The algorithms implemented in the system differ from those described in Section 6.2.
4. The solution suggested is only a bottom-up approach.

The benchmark is not a real-life problem and reducing the size of the images does not affect the accuracy of the measurement performed, moreover, the size of an image has no effect to the structure of the problem which is the major area of investigation of this chapter. Based on these reasons, the size of the images is reduced to match that of the CLIP7A system.

There are two reasons why floating-point is not used in the modified depth image. First, the rectangles in the depth image can well be separated within a 8-bit precision and secondly, the software developed for the CLIP7A system only includes integer operations. However, the implication of floating-point processing will be

considered in the design of the multi-layer system.

As demonstrated in Section 6.1, different approaches can be applied to solve a particular problem. Since the benchmark problem is only treated as a scene analysis problem, so different methods can be used to solve it. The algorithms implemented in the system attempt to exploit the parallelism afforded by the system. Additionally the algorithms map well onto the linear structure so that they are more effective and easier to implement. For the moment only a bottom-up solution is derived. A bottom-up solution also caters for other vision problems, such as Chinese character recognition and face profile recognition problem described in Section 6.1. In the problem under present review the bottom-up solution manages to provide a correct answer, nevertheless, the architectural requirements imposed by the top-down approach will be discussed in Chapter 9. The operation sequence of the modified problem is depicted in Fig. 6.4 and in the following sections the algorithms implemented in the CLIP7A system are described.

6.3.1 Low-level Processing

The goal of low-level processing is to locate all right-angle corners existing in both intensity and depth images. It is necessary to identify corners belonging to a single object and two processes are required to achieve this. First, all the connected objects are labelled so that they can be identified by a unique integer label and then a right-angle corner detection algorithm is applied to the labelled image. Corners belonging to different objects can be distinguished from their labels.

Labelling

Labelling is the assignation of a unique number (label) to each connected object - a collection of pixels having the same intensity. To design the labelling algorithm, it is important to consider how overlapped objects having different intensities can be labelled. It is desirable to first separate overlapped objects and to treat them independently. This can be achieved by thresholding using a carefully chosen level. However, it is necessary to consider all the overlapped pairs/tuples of rectangles appearing in the image and only one pair/tuple of the overlapped rectangles is processed at each iteration. To obtain the thresholding levels, each non-zero pixel (background has 0 intensity) receives intensity information from its four nearest neighbours and compares the incoming intensity values with its own. If the maximum value among the five pixels comes from a neighbour then a PE stores the maximum value otherwise a 0 is kept. The resultant image is made up of a few pixels representing the edges

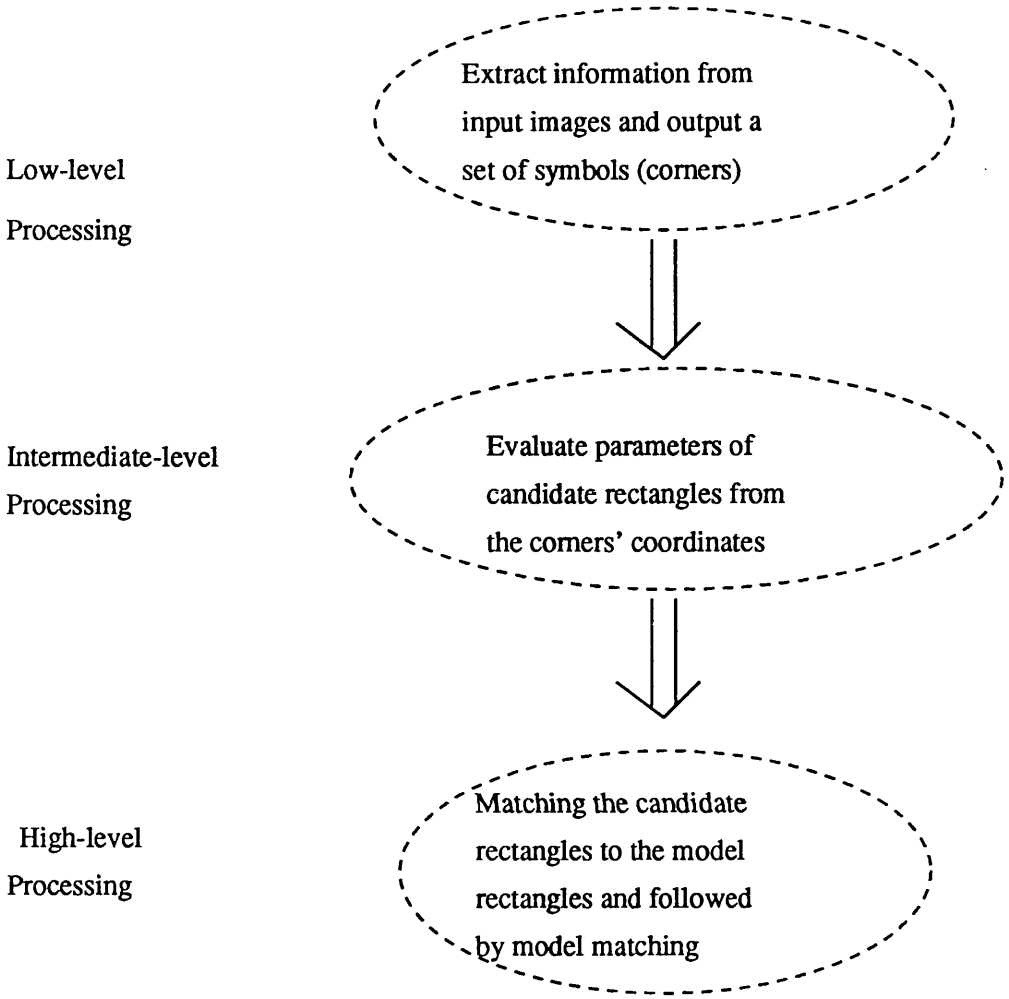
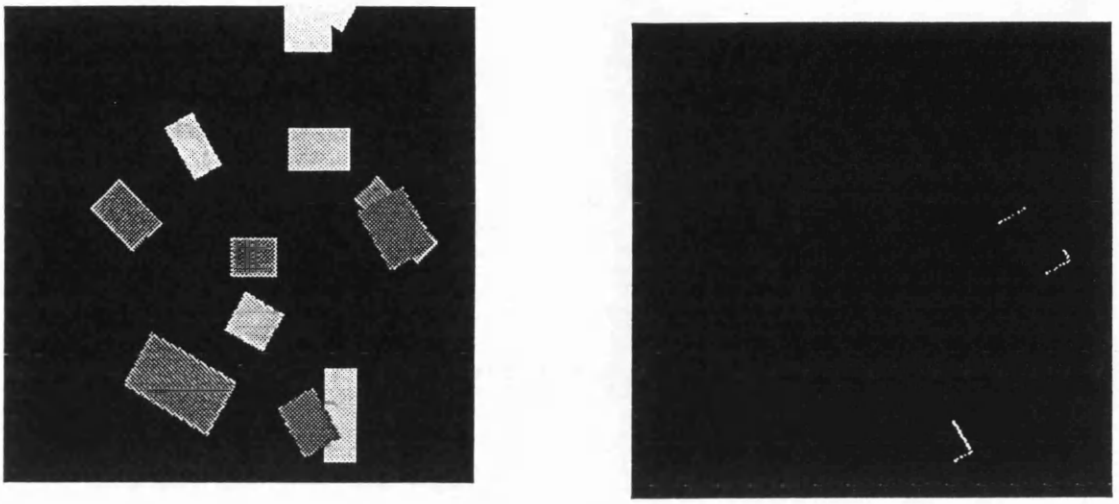


Figure 6.4 Generalised operation sequence of the modified benchmark problem

between overlapping rectangles, as shown in Fig. 6.5. Thresholding levels are determined from the values of those pixels and arranged in decreasing order. The values derived only cater for the overlapping rectangles and an extra value (1) is added so that all the remaining unlabelled objects can be labelled at the last iteration of the process. Thresholding is then applied to the input image with a different thresholding level at each operation. After each thresholding, the labelling algorithm is applied to the resultant image and labelled objects will be removed from the original intensity image, ensuring each object will only be labelled once.

After thresholding, a binary image is obtained so a binary labelling algorithm can be applied, this is developed exploiting the parallelism of the system. The labelling algorithm can be divided into two parts, the first part being to locate the objects and the second carrying out the labelling. The complete process can be divided into the following steps:

1. The system scans over the image and a vector, empty_v, is created to store information relating to the positions of empty and non-empty columns in the image. It is done by logical 'ORing' all the image vectors. In each non-empty region at least one object can be found.
2. Vector begin_row is also generated during the scan. Similar to empty_v, this vector has empty and non-empty regions. In each non-empty region, a constant is stored and the value of this constant is the row number where the first non-zero pixel in that region can be found.
3. A vector, label, is created. Similar to empty_v, vector label has both empty and non-empty regions, the positions of which are derived from empty_v. In label, each non-empty region carries a constant corresponding to a label value.
4. A bottom-up scan is carried out and vector end_row containing information of the row number where the last non-zero pixel in each column can be found is created. The input image is then divided into sub-windows by vector begin_row and end_row and there is at least one object within each sub-window.
5. All sub-windows are shifted to the top of the image (see Fig. 6.6(b)) and labelling begins by propagation.
6. Each sub-window receives a unique label value from vector label. The label value propagates from the top of an object. A non-zero pixel after receiving the label will store the label value and propagate the label to its nearest neighbours. If a zero pixel receives the label, it will not store the label and a '0' is propagated to its neighbours. This results in labelling the first object in each region as shown



(a) Input image

(b) Edges of overlapping rectangles

Figure 6.5 Locating the edges of overlapping rectangles

in Fig. 6.6(c).

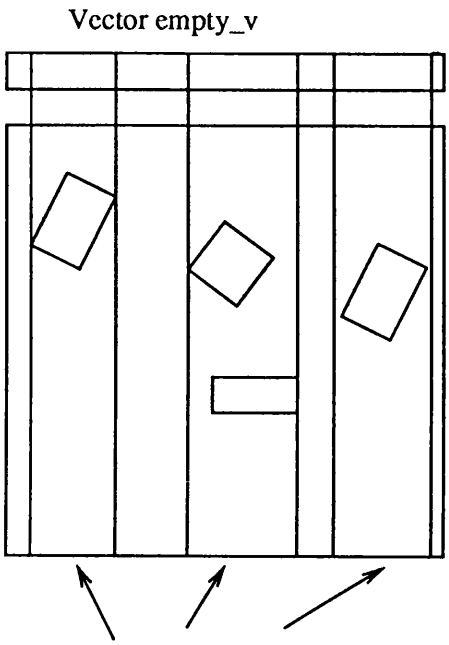
7. The labelled objects are removed from the input image (see Fig. 6.6(d)). Each non-empty region may contain more than one object and if the resultant image is not empty the process repeats from Step 1, otherwise the process ceases.

By repeating both thresholding and labelling algorithms, all the objects in the original intensity image will be labelled and a right-angle corner detection operation can proceed.

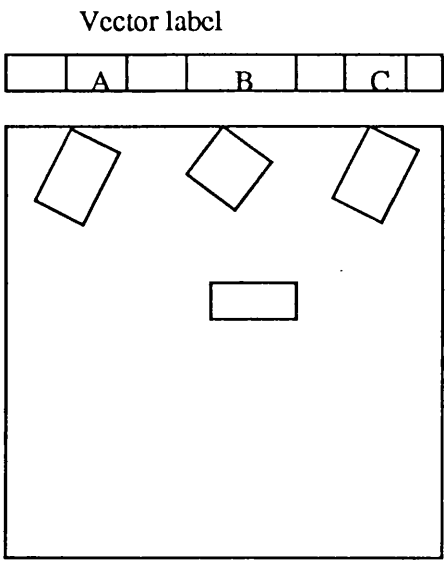
Right-Angle Corner Detection

The right-angle corner detection algorithm attempts to exploit the parallelism available in CLIP7A and involves local neighbourhood operations. In a 3x3 window, there are 8 patterns, as depicted in Fig. 6.7, which can be defined as a right-angle corners. The simplest method to identify a right-angle corner is obviously by template matching with the masks shown in Fig. 6.7, however, a 3x3 window cannot provide enough information to locate a right-angle corner. This is because the patterns shown in Fig. 6.7 may be part of many non right-angle figures. A corner is the intersection of two lines, in order to determine the property of a corner, it is necessary to consider both the corner and the line segments; this means that a bigger window is required. A 5x5 window appears to be an appropriate choice because it provides more information about the line segments extending from the corner point. To collect the information included in a 5x5 window demands extra inter-processor communication, moreover, memory required to store the different combinations that a 5x5 window can generate makes it very difficult to implement such an algorithm. The algorithm devised still uses the 3x3 window size but information is propagated from a pixel's neighbours and then accumulated. By repeating the operation twice, or three times, information collected by each pixel is equivalent to those covered by a window size of 7x7, or of 9x9.

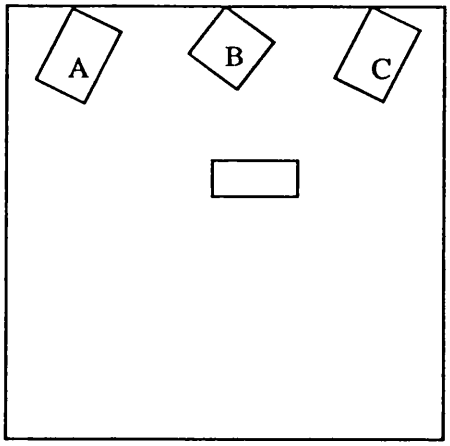
As briefly described above, a corner is formed by the intersection of two straight lines or edges. In a digitised image, a corner can be defined as a corner point plus the line segments extending from it. In a 3x3 window, it is possible to devise a set of masks representing the corners, line segments, and the patterns in between. These are shown in Fig. 6.7, Fig. 6.8 and Fig. 6.9. Masks 1 to 8, in Fig. 6.7, represent good right-angle corners whilst masks 9 to 12, in Fig. 6.8, are good straight line segments. Masks 13 to 20, in Fig. 6.9, are some of the patterns classified as weak right-angle corners, or straight line segments, and there are 34 masks in this category. Since there are only rectangles in the input images, it is possible to define a set of candidate right-angle corners that can be constructed from the masks described above. A good



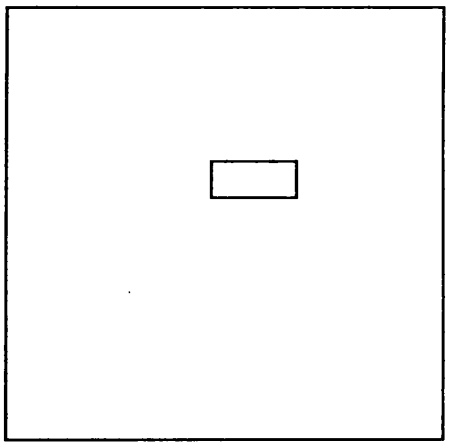
(a) Input image



(b) Objects shifted to the top of the image



(c) First object in each non-empty column labelled



(d) Labelled objects removed

Figure 6.6 The sequence of operations of the labelling algorithm

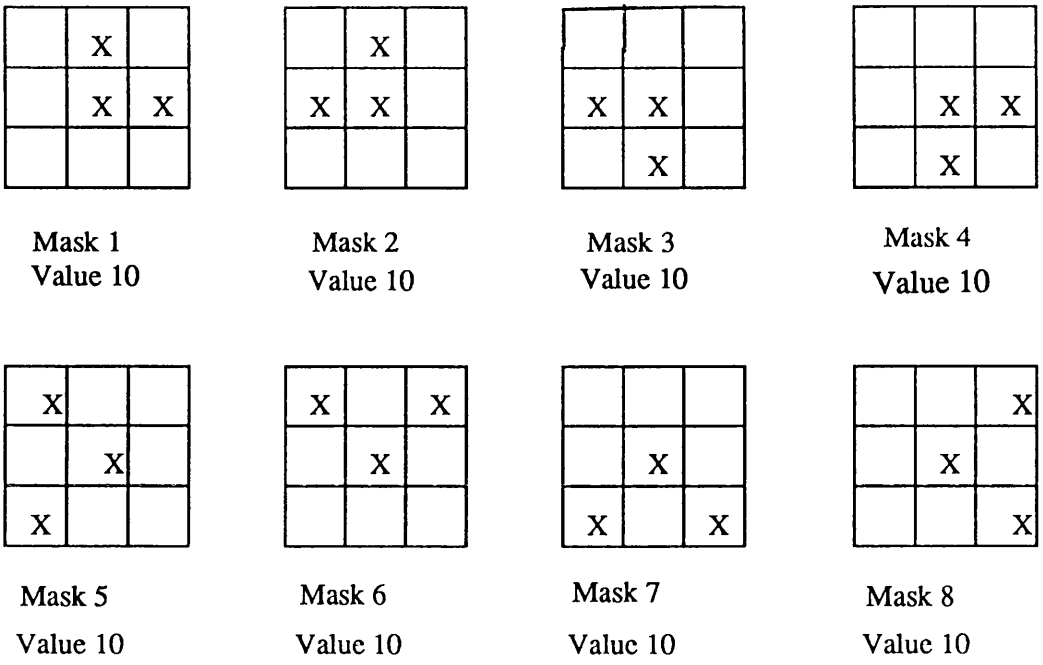


Figure 6.7 Good right-angle corner masks and their mask values

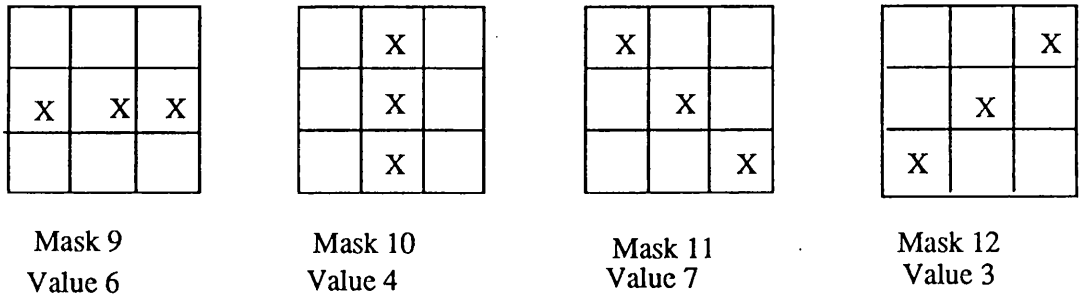


Figure 6.8 Good straight line segment masks and their mask values

right-angle corner, for example, should have Mask 4 as the corner and Masks 9 and 10 as straight line segments, as shown in Fig. 6.10(a).

A corner is formed by a set of masks and if the information conveyed by each mask can be propagated and accumulated then the property of a corner can be determined from the accumulated information. In order to achieve this, each mask is assigned a value, determined after a thorough study of the possible combinations. By accumulating mask values from a pixel's neighbours, the property of a corner can be determined and all the possible right-angle corners that can be extracted from the input images are represented by a set of discrete values (the accumulated results).

Each mask includes pixels within a 3x3 window and its value represents a pattern within that window. By adding the mask values belonging to a pixel's nearest neighbours together, this can be considered as gathering information regarding the property of an angle from a 5x5 window. This is because pixels covered by masks originating from the nearest neighbours are those forming the 5x5 window. After the first addition, each pixel accumulates mask values from its nearest neighbours. The accumulated result represents the property of a corner in a 5x5 window. If the resultant values now stored in a pixel's nearest neighbours are again collected then the information about the property of a corner within a 7x7 window is obtained. By accumulating the resultant values currently stored in a pixel's eight nearest neighbours, information representing the property of a corner within a 9x9 window is collected.

After a thorough study, right-angle corner patterns that can be obtained from the input images are divided into two groups by the pattern of the corner point. Right-angle corners having Masks 1 to 8 as their corner points belong to the first group and other possible right-angle corners (an example is shown in Fig. 6.10(b)) with other masks (e.g. Masks 14 and 15) as the corner points are classified as the second group. Corners belonging to the first group can be identified by gathering information within a 7x7 window, whilst a 9x9 window is required to correctly identify corners in the second group.

Since it is easier and more accurate to identify binary edges, the algorithm is performed after the thresholding operation described previously. The algorithm is divided into the following stages:

1. The edges of all the rectangles are obtained using a simple binary edge detector, as described in Chapter 5 Section 5.2.2.2.
2. An address is generated from a pixel's eight nearest neighbours, as described in Chapter 5 Section 5.2.1.2.

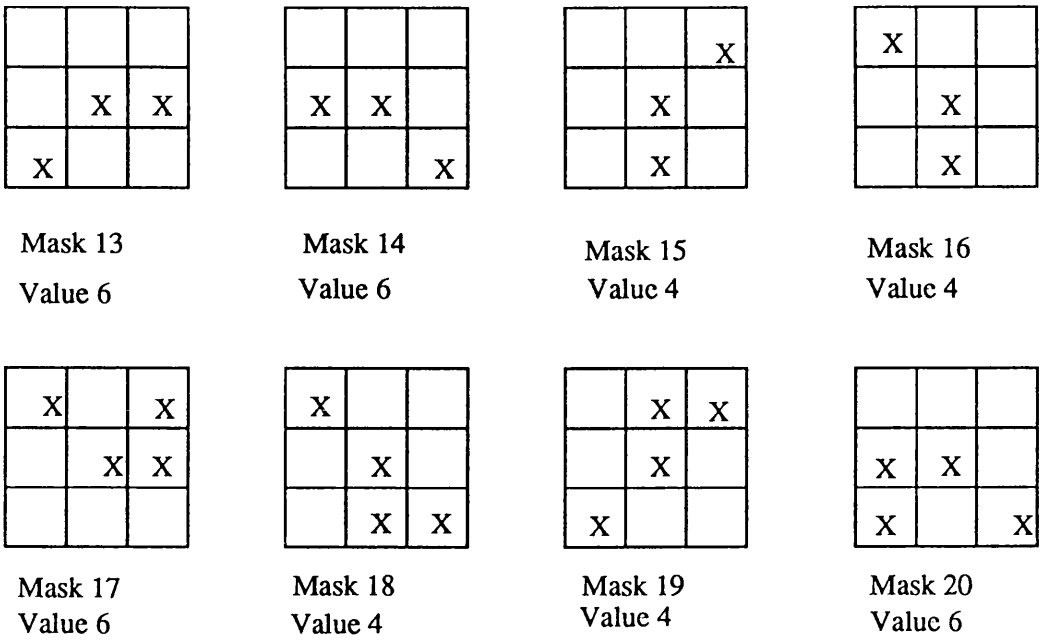
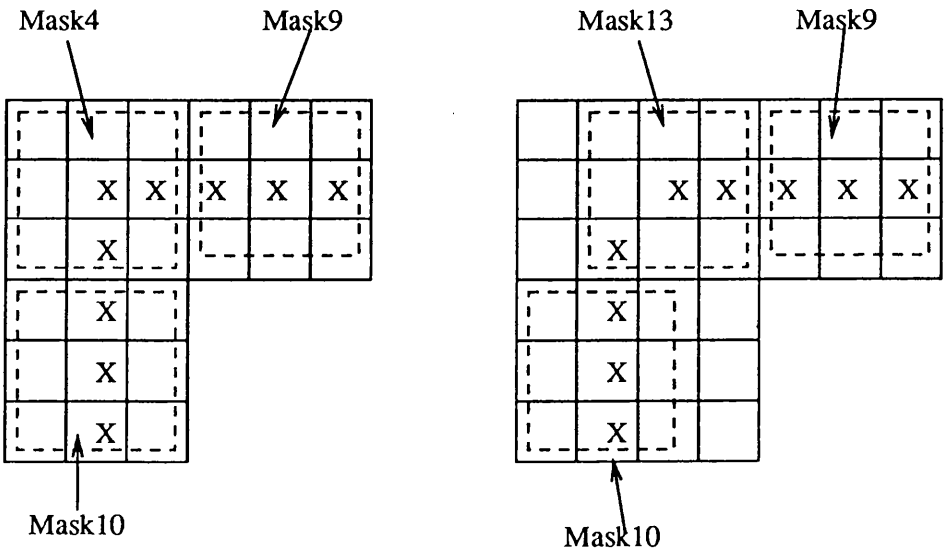


Figure 6.9 Masks represent neither a strong right-angle nor a straight line



(a) Right-angle corner in the first group

(b) Right-angle corner in the second group

Figure 6.10 Examples of right-angle corners

3. A lookup table is stored in each PE's local memory and the data stored are the mask values shown in Fig. 6.7, Fig. 6.8 and Fig. 6.9. The address generated in (2) is utilised to locate the corresponding mask value from the table.
4. Each pixel adds the mask values from its neighbours to itself and the total sum of the addition replaces the mask value.
5. The last operation is repeated.
6. Each pixel now has accumulated information provided by a 7x7 window. The sum stored in each pixel is used to address a second lookup table for the identification of corners in the first group. The data stored in the lookup table is either a 0 or 255. If 255, it represents a right-angle corner.
7. The operation in (4) is repeated and the accumulated value (the sum) represents the property of a corner within a 9x9 window. The sum is employed to address the third lookup table, in order to identify the second group corners. Only 0 and 255 are stored in the table, the value 255 symbolises the existence of a right-angle corner.
8. The result obtained from the algorithm is presented in the form of an image consisting of a few pixels with the value of 255 that represents the presence of a right-angle corner.

In Fig. 6.11, two examples are given to demonstrate the application of the algorithm. Because right-angle corners are represented by a set of discrete values, the two patterns shown in Fig. 6.11 can be distinguished according to their accumulated results.

If the number of right-angle corners located from an object is three or four then the object is regarded as a candidate rectangle and the parameters are computed from the X and Y coordinates of the corners. At this point the processing is concentrated on the intensity image but the same procedure including labelling and right-angle corner detection is applied to the depth image. In order to avoid the duplication of information provided by candidate rectangles appearing in both depth and intensity images, candidate rectangles extracted from the intensity image are used as a mask to filter any redundant information from the depth image. Candidate rectangles extracted from the depth image are added to those obtained from the intensity image and this completes the low-level processing.

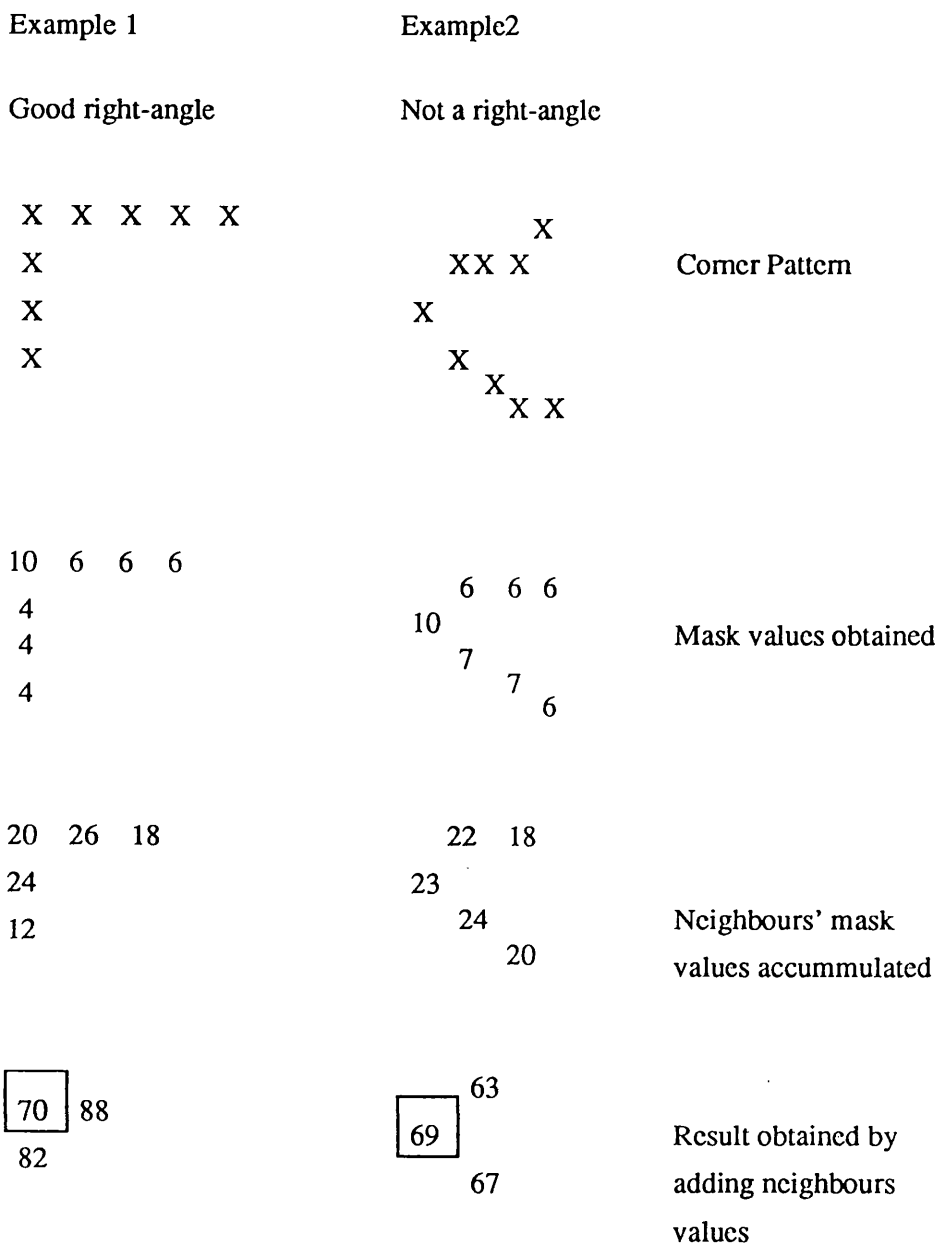


Figure 6.11 Examples to demonstrate the right-angle corner detection algorithm

6.3.2 Intermediate-level Processing

Intermediate-level processing involves the evaluation of parameters of candidate rectangles (including the centre coordinates and length of the axes) from the corner points located by the low-level processing operations. An object with 3 or 4 right-angle corners detected is regarded as a candidate rectangle. Each candidate rectangle is given a unique candidate number starting from 1. The five parameters listed in Table 6.2 are determined by the triangle that is defined by the three successive corners shown in Fig. 6.12. The processing unit employed by the CLIP7A system is ineffective at evaluating the square-root function required to compute the length of the axes and the coordinates of the corners are passed to the host to have the parameters evaluated. The results are transferred back to the system for high-level processing. The parameters of each candidate rectangle are stored in every PE's local memory. Each PE carries all the parametric values evaluated and the values are arranged in the form of lookup tables as shown in Fig. 6.13. Such an arrangement enables each PE to retrieve the parametric values of any candidate rectangle rather easily by a local addressing operation, utilising the candidate number as the low-byte address. This property will assist the implementation of the high-level operations.

6.3.3 High-level Processing

At this stage all the possible candidate rectangles have been extracted from both intensity and depth images and the rest of the processing will not involve any pixel based operations. In the high-level processing, two tasks are performed namely: the matching of candidate rectangles to the model rectangles and the determining of the best matched model by the rectangles' spatial relationships. The results of the matching of candidate rectangles and model rectangles are expressed in the form of a table called the compatible table. The compatible table will be utilised in the model matching process. The main challenge is to exploit the parallelism provided by the CLIP7A system. The data type employed now is not in image format and there is no natural match between the data type and the system. The data type at this stage is rectangle parameters which can be regarded as a vector with five elements. Since each candidate rectangle has to be compared to individual model rectangles, parallelism can be applied to the comparison. If each PE represents one candidate rectangle or model rectangle then a total of 256 comparisons can be made simultaneously. Similarly, if each PE represents a rectangle parameter then a vector can hold data of 64 rectangles and comparing two such vectors is equivalent to the comparison of 64 rectangles.

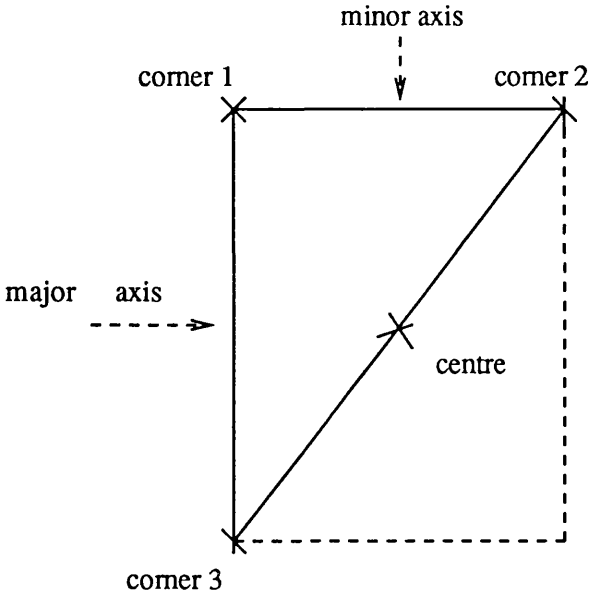


Figure 6.12 Rectangle parameters defined by 3 contiguous corners

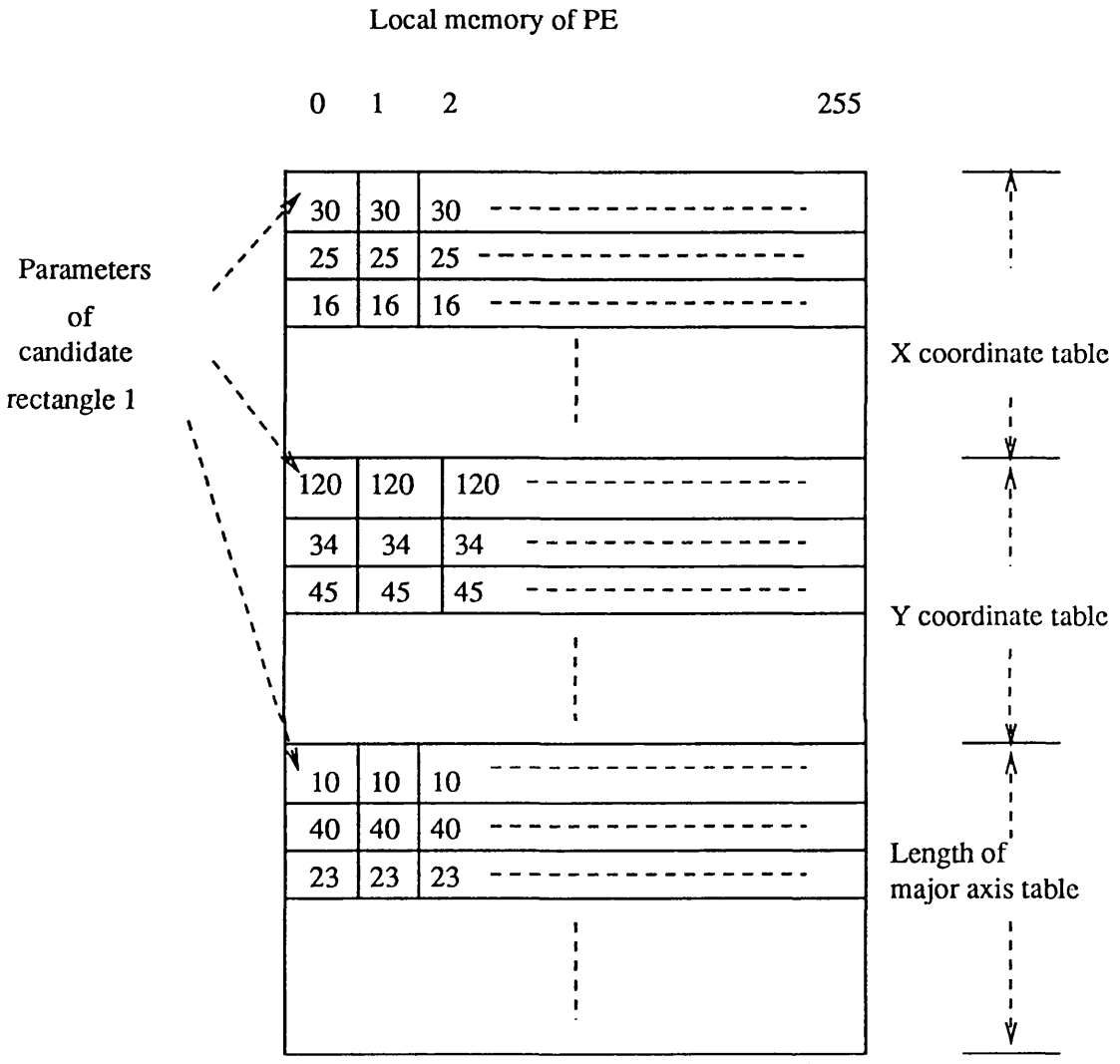


Figure 6.13 Structure of lookup tables for candidate rectangles' parameters

The Compatible Table

Each model has its own compatible table which provides information about the candidate rectangles compatible to its model rectangles. A candidate rectangle is compatible to a model rectangle if its major axis, minor axis, intensity, and depth are equal to, or within a pre-defined error margins, to those of a model rectangle. As described in the previous Section, each candidate rectangle is given a unique candidate number which can be used as an index to the memory locations where the candidate rectangle's parametric values are stored. This means the compatible table will only contain the candidate rectangle numbers. The number of columns occupied by each compatible table is equal to the number of rows available in the model's relational table, whilst each table has the same number of rows. Each compatible table is sub-divided into five sections, this is because each model rectangle can be compatible to several candidate rectangles, up to five in this arrangement. Information about candidate rectangles compatible to the same model rectangle will be stored in different sections of the compatible table.

In order to create the compatible table, the physical parameters of each candidate rectangle must be compared to all model rectangles. If there are N candidate rectangles and M model rectangles then it takes 4*M*N comparisons to form the compatible table but it is possible to parallelise the operation by arranging a PE to represent one parameter. In the test data, there are only 7 models with a total of 61 model rectangles so there are 244 parametric values to be considered. Two vectors, one for the depth and the other for the remaining parameters, are sufficient to carry all the model rectangles' parametric information. On the other hand, each candidate rectangle is assigned two vectors, one for the intensity and length of the axes, whilst the depth value occupies the other. The parameters in both vectors of a candidate rectangle are repeated along the vector as shown in Fig. 6.14. By doing this, a comparison between model vectors and candidate vectors is equivalent to comparing a candidate rectangle with all the model rectangles at the same instant and the total number of comparisons required is reduced to 2*N. The complete process is described in the following:

1. All the model rectangles' parameters are read and their parameters are divided into two groups, as described above, and arranged in two separate vectors.
2. Each candidate rectangle occupies two vectors and the data is arranged in the format shown in Fig. 6.14.
3. Two error margin vectors, EV1 and EV2, using the same format as the candidate rectangle vectors are formed. EV1 is for the intensity and the length of the axes for which the margin are 0 and 4 respectively. EV2 is for the depth value and the

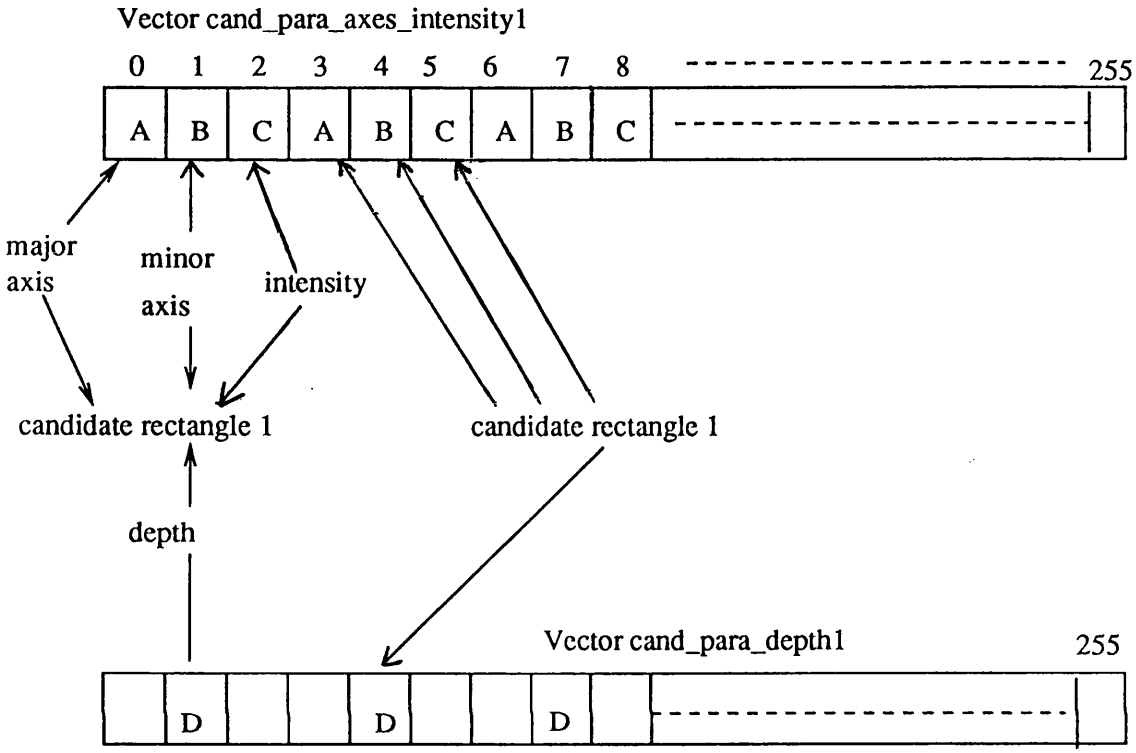


Figure 6.14 Formats employed to represent paramcters of a candidate rectangle

error margin is 2.

4. The intensity and length of the axes of candidate rectangle 1 are first compared with the model rectangles and the results tested by the error margin. A flag is set if an item lies within the margin and if all three parameters are compatible then three flags should be generated in three consecutive PEs. A neighbourhood operation is applied to gather the results of those flags and a collective result is generated. The collective result is positive only if all three flags are set and the collective results are stored in a vector, CR_V, using the format of the depth vector for the candidate rectangles.
5. The depth data between candidate rectangle 1 and the model rectangles is compared and the results checked with the error margin. As above, a flag is set if the two values are compatible. All flags are stored in a single vector, DR_V.
6. The vector CR_V generated in (4), and vector DR_V from (5) are combined by a logical 'AND' operation and the results held in vector CDR_V.
7. The vector CDR_V is examined and a positive result represents the compatibility of the candidate rectangle to a model rectangle, determined from the position of the positive result in the vector (CDR_V). The candidate rectangle number is then stored in the corresponding model's compatible table.
8. The sequence from (4) to (7) is repeated, but with different candidate rectangles, until all candidate rectangles have been tested.

The structure of the compatible tables is shown in Fig. 6.15. Where it can be seen that model 1 model rectangle 0 is compatible with candidate rectangle 4 while model 2 model rectangle 0 is compatible to candidate rectangle 6. It should be noted that the candidate rectangle number begins at 1, so 0 stands for an empty entry. The compatible tables are stored in the local memory of the linear array. Once the compatible tables are created, relational data matching can begin.

Relational Data Matching

A model is constructed from model rectangles related to each other according to their distances and angles of bearing. The spatial relationships between model rectangles are described by a relational table, as shown in Table 6.1. In order to match the input scene to a model, the spatial relationships between candidate rectangles compatible to model rectangles must equal (or within the allowed margins) the values provided by the relational table. Each row in a relational table defines the spatial relationships between two particular model rectangles which are specified by their

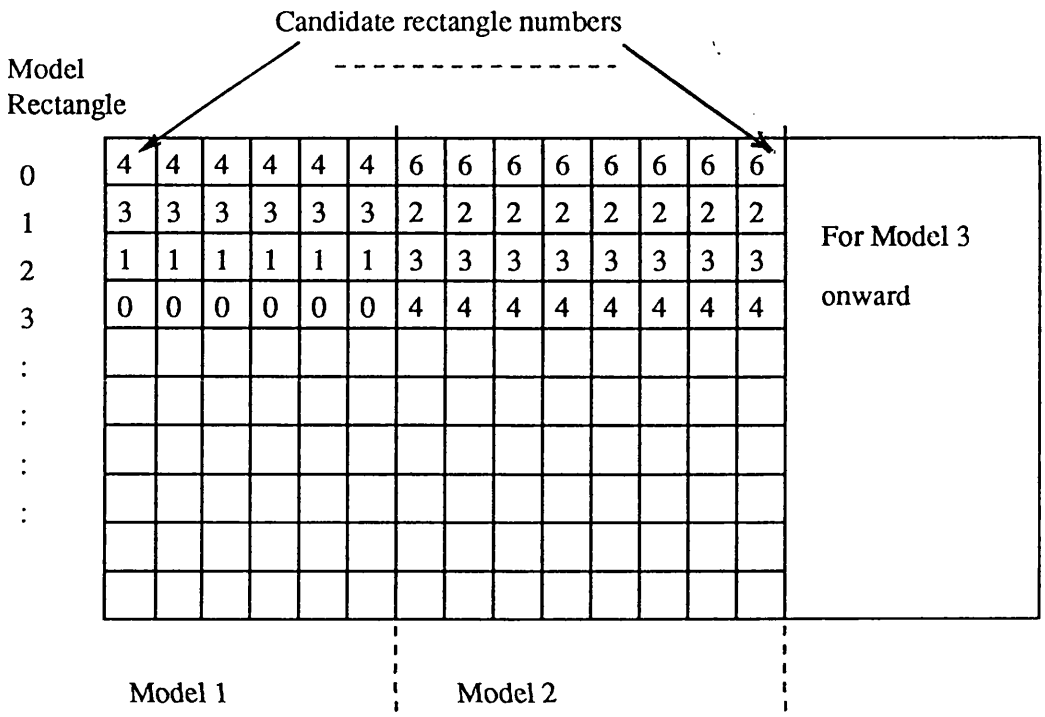
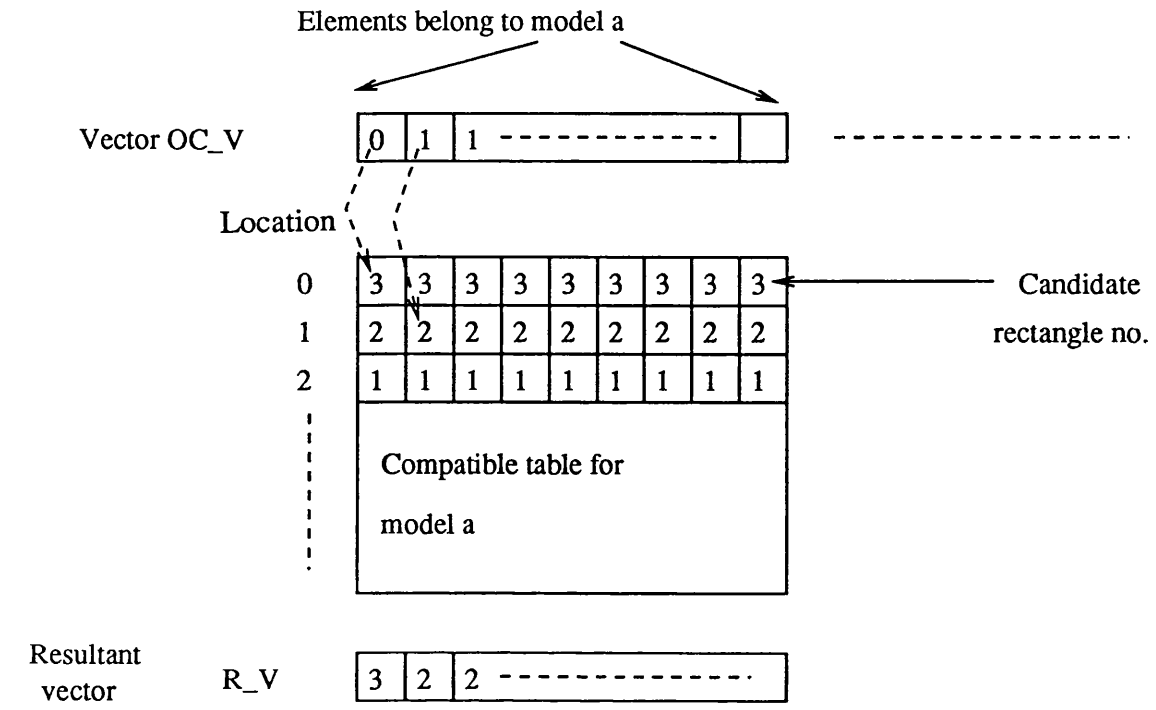


Figure 6.15 The structure of the compatible tables

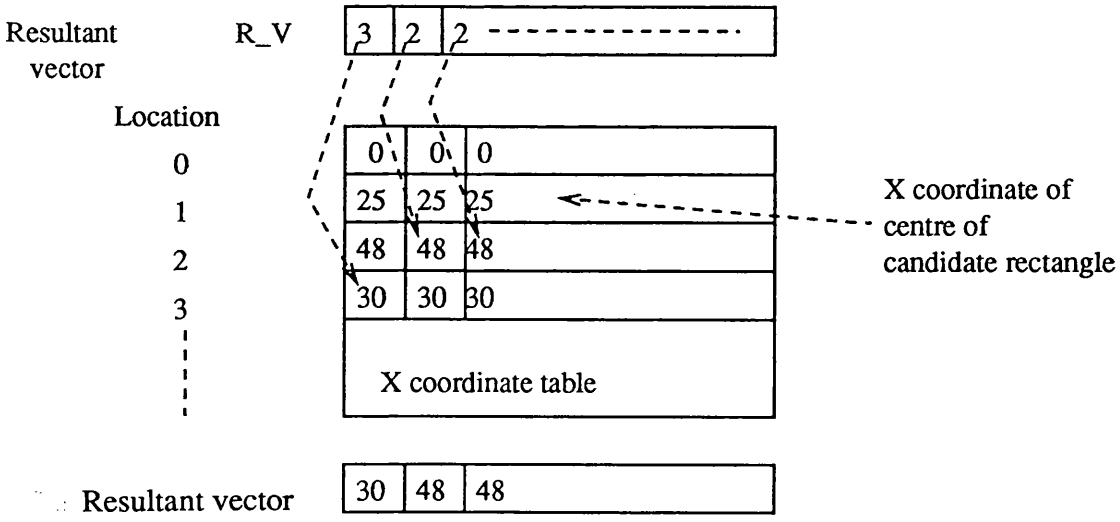
numbers. Each row is divided into four columns: origin, destination, bearing, and distance. The definitions of the names of the columns are depicted in Fig. 6.3. The matching process requires, initially, the identification of candidate rectangles compatible with the model rectangles specified in the origin and destination columns. The distance and bearing between the candidate rectangles must then be evaluated and finally the computed results compared with those provided by the relational table. If the computed values are within a certain error limit then a link is said to be formed between the two candidate rectangles. The maximum number of links that can be formed for each model is equal to the number of rows in the relational table. The total number of links formed by the candidate rectangles with respect to the relational table reflects the likelihood of the model being present in the input scene. The percentage of links formed is evaluated and the best matched model will be that with the highest percentage.

The matching process involves the searching of the compatible table in order to locate the compatible candidate rectangles followed by the evaluation of the spatial relationship between them. These values are compared with those in the relational table with error margins taken into consideration. As in the previous section, parallelism is a major factor to be considered in the design of the matching process. This time parallelism is applied to both the searching of the compatible table and the comparison process. The relational table (see Table 6.1) is divided into four columns. The first (origin) and second (destination) column both contain the model rectangle numbers from which the corresponding compatible candidate rectangles must be extracted. Both origin and destination columns are used as indices to address the compatible tables. The origin column in each relational table is combined and a vector (OC_V) formed. This maps well to the compatible table because the width of a compatible table is equal to the height of the corresponding relational table. The vector, OC_V, is then used to reference the compatible tables by local addressing, as shown in Fig. 6.16(a). The result is a vector, R_V, containing all the candidate rectangle numbers that are compatible with the model rectangles in the origin column. The vector R_V is then used to locate the X and Y coordinates of the candidate rectangles via local addressing, as shown in Fig. 6.16(b). The same procedure is applied to the destination column. The complete process is summarised below.

1. Information is extracted from the relational tables and four vectors are employed to store the data representing the origin, destination, distance, and bearing.
2. The origin vector (OC_V) is used as a pointer to the compatible tables from which candidate rectangle numbers compatible with the model rectangles are



(a) Retrieving candidate rectangle number from the compatible table



(b) Retrieving X coordinate of centres of candidate rectangle

Figure 6.16 Addressing the compatible table and retrieving X coordinates of the centres of the compatible rectangles

extracted and stored in vector R_V.

3. Vector R_V is used as an index to address the coordinate tables, so that the X and Y coordinates of the centres of compatible rectangles are retrieved from the memory.
4. The centre coordinates for the destination rectangles are extracted by repeating (2) and (3) with the origin vector replaced by the destination vector.
5. The distance and bearing of the origin-destination pair are computed (the computation is performed in the host for the reason stated in Section 6.3.2.)
6. Computed distance and bearing are arranged in two separated vectors which are then transferred back into the CLIP7A system. As stated in (1), values regarding both distance and bearing in the relational tables are arranged in two vectors as well. A vector comparison will accomplish all necessary comparisons between the computed and given values of either the distance or bearing.
7. The vectors of the distance are compared. If any differences between the computed values and the model data are within the error margin a flag is set. The flags are stored in a vector. Similarly, the vectors of the bearing are compared and a second vector carrying all the flags is created. The two flag vectors are combined by a logical 'AND' and presented in a single vector. Each flag in the combined vector represents the formation of a link between two candidate rectangles.
8. As one model rectangle can be compatible with several candidate rectangles and each model has five sections of compatible table. Steps (2) to (6) are repeated if the compatible table has more than one non-empty section. The number of flags established at each section forms a cumulative total.
9. As each flag represents a link between two candidate rectangles, the percentage of the number of links formed is evaluated, with the matched model having the maximum percentage.

6.4 Conclusion

In this Chapter the algorithms implemented in the CLIP7A system to solve the modified DARPA benchmark problem have been described. The purpose of the exercise is not the benchmarking of the CLIP7A system but rather to collect information concerning the design of a multi-layer system governed by a vision problem. The modified problem embodies processing operations at low, intermediate, and high levels. At low-level processing, the operations are devised to extract the right-angle

deficiencies:-

floating point

trigonometric functions, $\sqrt{}$

corners existing in the input image. A rectangle's parameters are evaluated from the coordinates of its corners during intermediate-level processing. At the high-level stage, two matching operations are involved. The first, to match the rectangles extracted from the input image to the model rectangles; the second to match the spatial relationships between the rectangles in order to identify the input scene from the model scenes. The implementation of the algorithms establishes three points concerning the design of an image processor:

1. The structure of the problem

   The modified benchmark problem and other examples given in Section 6.1 are three-level vision problems. The solution proposed employs a bottom-up processing sequence, i.e. low-level processing followed by intermediate and then high-level processing. Implementation of the algorithms established the fact that parallelism can be applied to each level through matching the data format to the structure of the system. For example, parallelism is achieved in the matching process by arranging the data into a vector format which maps well onto the linear structure. It also confirms that the amount of data to be processed decreases as the processing level increases. In the benchmark problem it changes from the size of the image to the number of corners and finally the number of rectangles so a reduction in the number of processing elements employed, from the base layer upwards, is considered.

2. The processing requirements

   In solving the benchmark problem, there are two occasions when the processing power of the CLIP7A system is inadequate to perform the required computation. The first is the floating-point operation demanded in the processing of the original depth image. The second is the trigonometric function and square-root function required in evaluating the parameters of a rectangle. Either software or hardware can resolve the problem and choice between the two rests on the cost-performance ratio that is affordable or requested by the designer. More discussion of this issue will be included in Chapter 8.

3. The communication requirement

   A communication requirement is created by the transformation of data between different levels of processing. Between low and intermediate levels, the corners belonging to a single rectangle need to be grouped together in order to evaluate the parameters. From intermediate to high-level, the rectangles (represented by

their parameters) are further organised into a scene for matching. If the parallelism available at each level is one PE for a corner, a rectangle, and a model respectively then a PE representing a rectangle only needs to communicate with three or four PEs which stand for the corners. However, each model PE must be able to communicate with all the rectangle PEs in order to create an overall description of the input scene. An example communication network for a two rectangle two model scenario is depicted in Fig. 6.17. This demonstrates how the inter-layer communication network is affected by the data structure employed at each level. The example is far from complete and a detailed study of inter-layer communication will be included in Chapter 7.

After the study of architecture requirements related to the structure of a problem, together with the design criteria for linear arrays derived in Chapter 5, the design of a multi-layer system can proceed and this will be covered in the following two chapters.

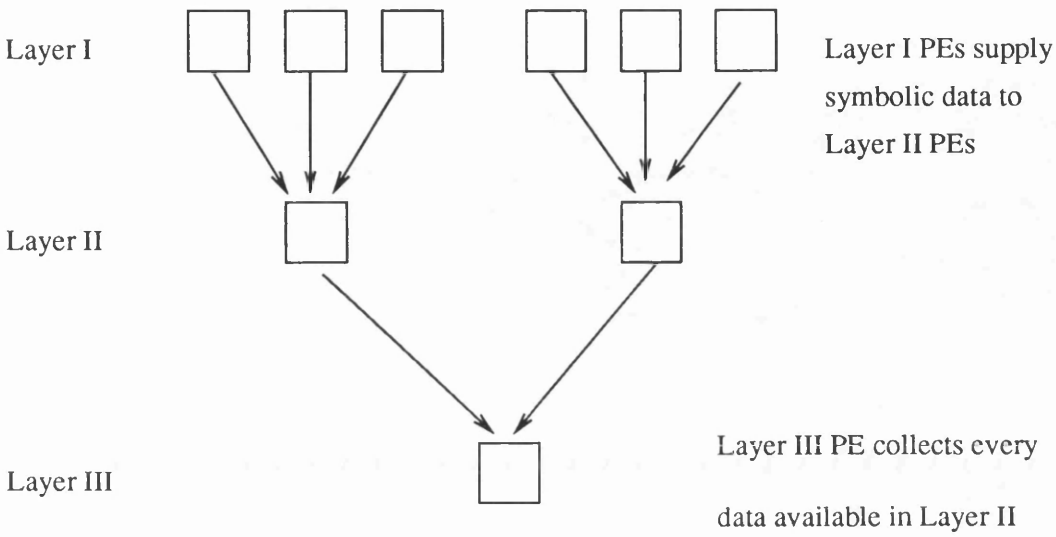


Figure 6.17 An example inter-layer connection network

Lacko delzi.

Chapter 7
The Design of a Multi-layer System

7.1 Introduction

In the last two chapters, four issues relating to the design of a multi-layer system for image processing were examined, namely,

1. The structural properties of linear arrays;
2. The effectiveness and applications of local autonomy;
3. Architectural issues resulting from an integrated vision problem; and
4. Processing requirements at different levels of operations.

In this chapter, results obtained from investigation of the above will be applied to the design of a multi-layer system for image processing which will handle vision problems mapped well to the structure depicted in Fig. 6.1. The examples given in Chapter 6 Section 6.1 reflect the diversity of problems being covered. As the multi-layer system is intended to handle a wide range of problems, it may not be optimised for a particular problem.

The proposed multi-layer system will include three major components: a host computer, video I/O devices and the multi-layer unit, as shown in Fig. 7.1. Since the multi-layer unit is the principal concern of the present work, only a brief description of other components is given. The host computer serves as an interface between the multi-layer system and its users. A user will develop application programmes in the host, which must consequently provide proper software facilities for the task. The host computer will also provide massive data storage, such as hard disk, for images and user programmes. Choice of host computer depends on the working environment of the proposed system. The video I/O devices may consist of different image input units, including video cameras or a video player for recorded information, and display units, such as TV monitors.

The design of a multi-layer unit involves initial consideration of its overall structure. Since the basic structure to be used in the system is a linear array, the overall structure of the system depends on the number of layers employed and how elements in different layers are connected.

7.2 General Structure of the Multi-layer Unit

The general structure of the multi-layer unit, depicted in Fig. 7.2, can be divided into eight components, including three linear arrays, two inter-layer connection

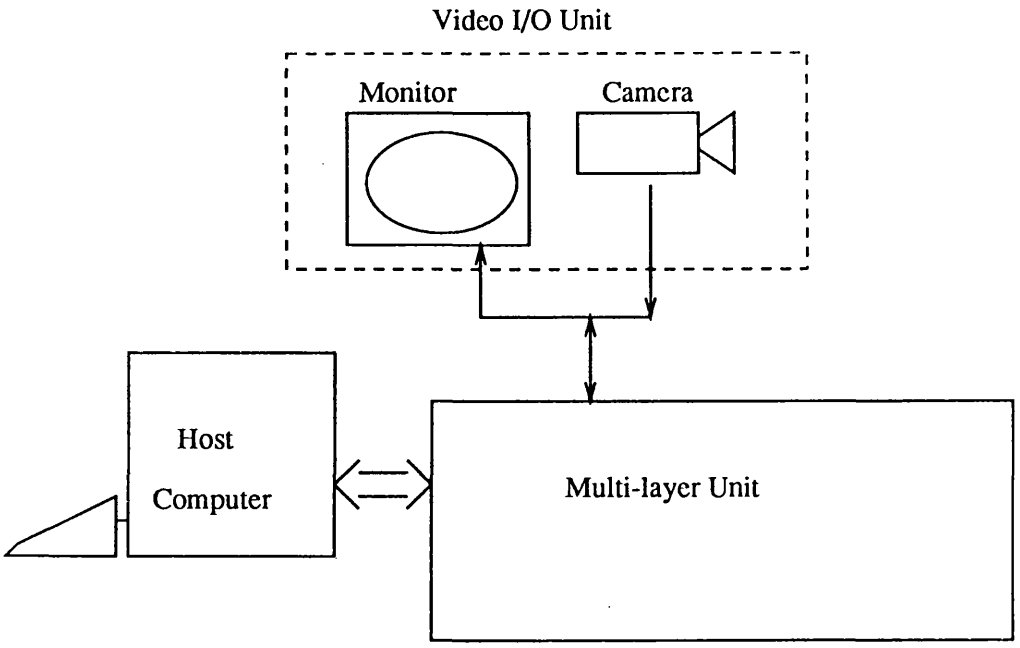


Figure 7.1 An overview of the multi-layer system

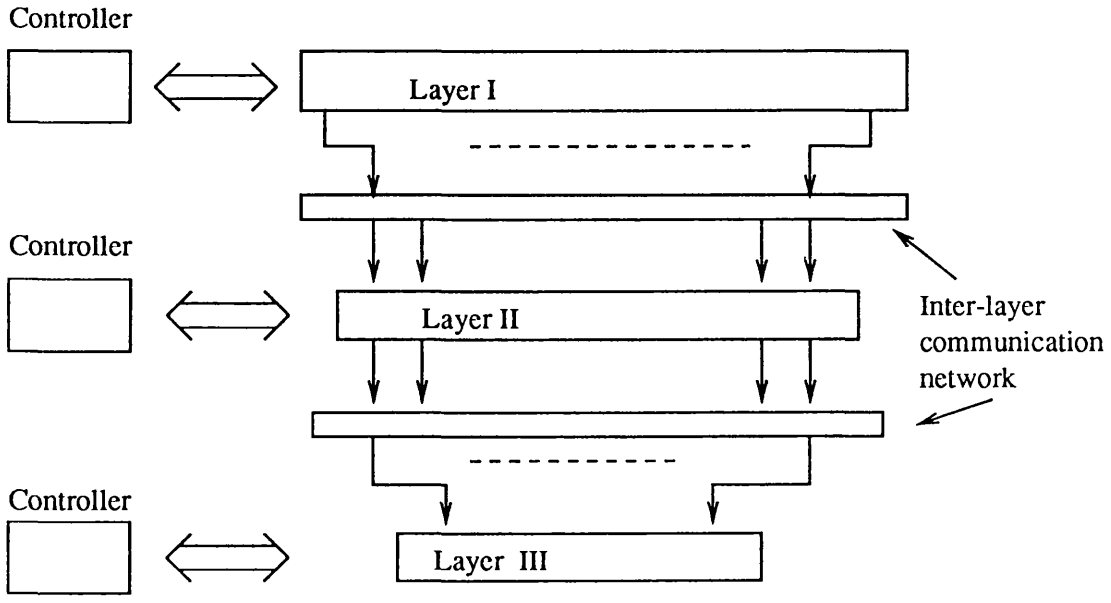


Figure 7.2 Block diagram of the multi-layer unit

networks and three controllers. In this section the number of processing elements in each of the linear arrays is determined.

The number of processing elements employed in each layer is governed by the quantity of data to be processed. In layer I, data is the pixel values of the input image so the quantity of data is equal to the size of an input image. There are, however, many different resolutions being utilised in various systems. For the proposed system, input images are confined to 256x256 pixels, each carrying 8-bit of information. The layer I array embodies 256 elements, each of which will process pixels in the same image column. As demonstrated by the CLIP7A system, the one-PE-per-image column arrangement provides a simple programming model and control requirements.

In the proposed arrangement, an input image is divided into 256 rows of image vectors to be processed one at a time. In many situations, a single operation has to be applied to each image vector, meaning that each operation must be repeated 256 times. The basic control requirement is a repeat mechanism which can be achieved rather easily. A further discussion of the controlling mechanism and the programming model is found in Chapter 8.

In layer I, the data type to be processed is image, which is represented by a consistent amount of data, i.e. 65536 8-bit values. After low-level operations, information regarding the input image is conveyed to the next layer via groups of data which are regarded as symbols. As described in Chapter 6 Section 6.1, a symbol can represent different entities in different problems and the number of symbols involved may vary from image to image. Information carried by a group of symbols represents an object or pattern in the input image and the function of the layer II PEs is to operate on the symbolic data belonging to the same object or pattern in order to generate a meaningful description of that object or pattern. This consideration suggested that the number of elements employed in layer II should equal the number of objects appearing in the input image, but as the number of objects present in each image is unpredictable it is, therefore, difficult to determine the exact number of PEs to be used.

As concluded in Chapter 6 Section 6.4, the number of objects present in an image is less than the number of image pixels implying that a reduction in the number of PEs available in layer II is appropriate. With reference to the examples given in Chapter 6 Section 6.1, there are, roughly speaking, three types of images, differentiated according to the number of objects present. First, those images with a single object which might be either a face profile or a Chinese character; secondly, those with a hundred or more objects - the electrophoresis gel images; thirdly, those found in the benchmark problem where the number of objects ranges from ten to twenty.

The number of PEs employed in layer II must, therefore, be able to adapt to process the different types of images. The factor of cost must also be considered. Based on these requirements, 16 PEs will be provided in the layer II array. With 16 elements, the layer can be easily adapted to the benchmark type images which also represent images in other scene analysis problems. If there are more than 16 objects in the input image then some elements will operate on more than one set of symbolic data. In Chapter 9 a discussion regarding how to apply such a system to process an image with only one object is given. When 16 PEs are insufficient to handle the computing power demanded by certain problems then expansion of the array must be considered with this supported by the inter-layer communication network. The cost of 16 PEs is moderate and this satisfies the cost requirement.

In the final layer, the function of the PEs is to match the input image to a set of models or database. Usually there is a single set of models or a single database, but the amount of data contained in them may be enormous. Instead of using a single processing element, the database or model set is split into sections, each of which will be searched or matched by an individual element simultaneously. The number of sections that a database, or a model set, should be divided into depends on the size of the database. In the current design, a database is divided into four sections, this implying the employment of four processing elements. Once again extra elements will be provided if four elements prove to be insufficient and the inter-layer communication network must consequently be capable of supporting such a requirement.

The number of processing elements employed in each layer has been suggested, based on the quantity of data to be processed by, and the cost of, each layer. Communication between elements in different layers is considered in the following sections but one criterion which those communication networks must be able to fulfill is to enable the linear arrays in layer II and layer III to be expanded easily. Efficiency and fault tolerance of the network are also considered.

7.3 Inter-layer Communication Networks

In the previous section, the general structure of the multi-layer unit was described. This section considers the design of the inter-layer communication networks, whilst the implementation of those networks is considered in Chapter 8. Major design criteria for the networks included are presented below.

Efficiency: passing data between elements in different layers must be efficient, so that the time required for inter-layer communication is minimal in comparison with the processing time required to solve the problem.

Fault tolerance: when an element fails in any layer it should be bypassed and replaced by a functioning element in order to retain the parallelism or computing power. There are two problems related to the bypassing of a faulty element and the connecting of a new element. The first is retaining or re-connecting the communication links between the faulty element and elements in other layers. The second that new communication links must be established between the new element and elements in other layer. Both hardware and software are required in order to achieve fault tolerance and an optimum design will minimise requirements of both.

Expandability: this applies mainly to layer II and layer III. As discussed in the previous section, the number of PEs employed may not match well to the quantity of data evolved at those levels. In some situations, where the number of PEs at layers II and III is insufficient to perform the operations effectively, the addition of extra elements to the array in order to improve its parallelism may be necessary. Expanding a single layer requires new communication links between additional elements and elements in other layers and this, in return, may affect existing communication links. The communication network must be flexible, so that the addition of new elements has a minimal effect on existing communication links.

In the following sections, the design of inter-layer communication networks based on the above criteria will be described.

7.3.1 Different Connection Methods

Two inter-layer connection networks are considered here, the first connects elements in layers I and II, whilst the second connects elements in layers II and III. Before discussing these particular networks a general description of connection networks applicable to two linear arrays is given. Five common connection methodologies (see Fig. 7.3) are presented below.

1. One to one mapping: the number of processing elements in each layer is the same and each processing element in the first layer is connected to a unique element in the second layer.

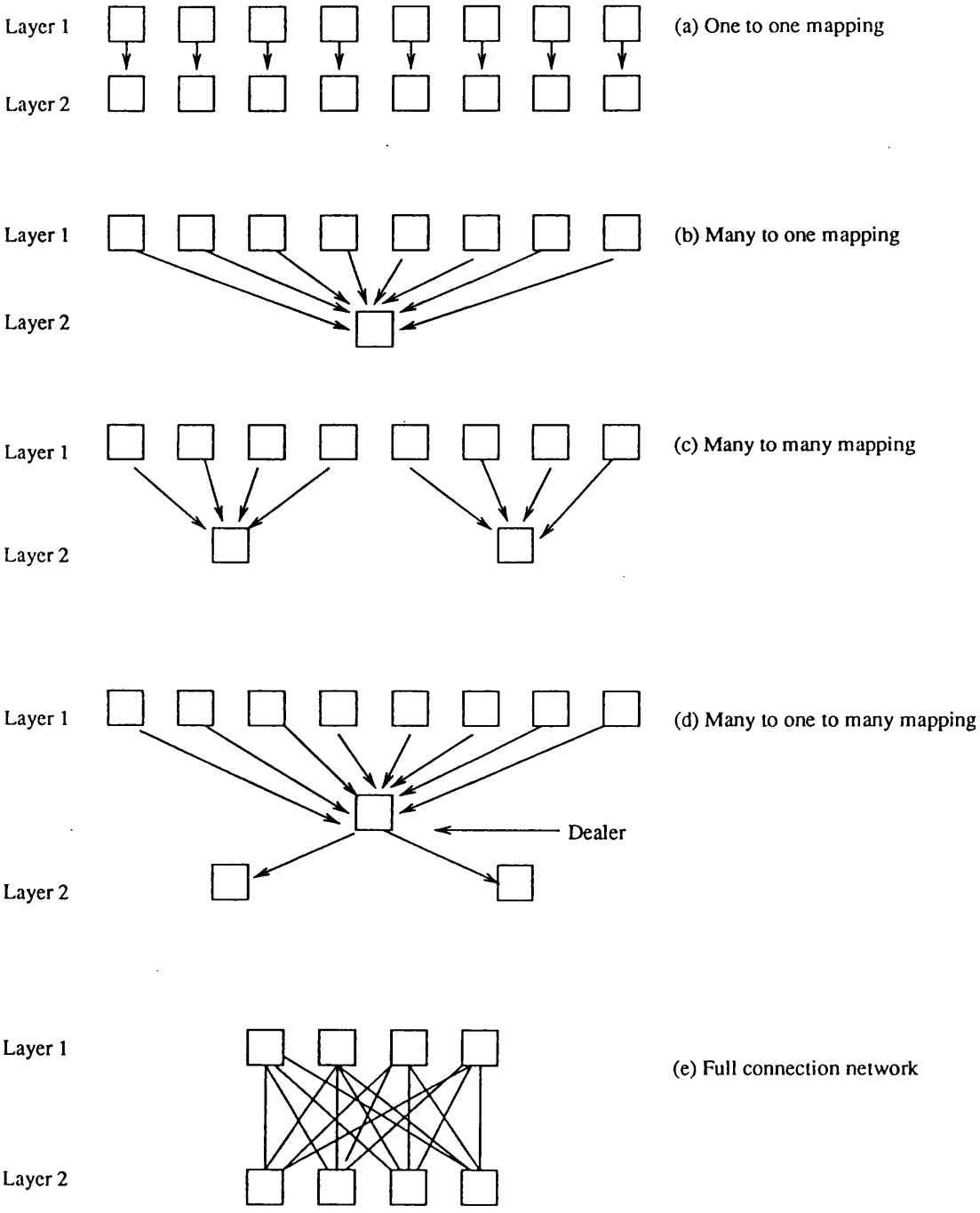


Figure 7.3 Different connection networks

2. Many to one mapping: this employs a single processing element in the second layer to collect all the information from the first layer and is the most straightforward method.
3. Many to many mapping: processing elements in the first layer are grouped together with elements in the same group being connected to the same element in the second layer.
4. Many to one to many mapping: the information coming from the first layer is collected initially by a single element (called the dealer) and then distributed to the processing elements beneath it.
5. Full connection network: each element in the first layer is connected to all elements in the second layer so that if there are M elements in the first layer and N elements in the second layer, a total of MxN communication links are required.

There are, certainly, many other ways that processors in two layers can be connected and one way to describe those connection networks is by an interconnection function f [65]. PE i in the first layer is connected to PE $f(i)$ in the second layer, where i represents the position of a PE in a layer. The one to one mapping, for example, can be described by the interconnection function $f(i)=i$. After introducing the different connection networks, the next step in the design procedure is to select the most suitable networks for the multi-layer unit.

7.3.2 The Layer I and Layer II Communication Network

As described above, the design of a communication network must take into account the factors of efficiency, fault tolerance and expandability. An effective communication network should map effectively to the structure of the problem, which is governed by the relationships between the data types evolved at each level.

In the proposed multi-layer system, the number of elements employed in each layer has already been determined and consequently, the one to one mapping and the many to one mapping cannot be applied. The full connection network can be applied to all situations but has the major disadvantage of high cost due to the large number of communication links required. In the proposed system, there are 256 PEs in layer I and 16 PEs in layer II so 4096 communication links would be required to construct a full connection network. The problem of contention also makes the controlling mechanism for the network quite complex. Contention arises when two or more elements attempt to communicate with the same element and a complex controlling

mechanism is required to ensure that only one layer I element is communicating with any layer II element. The complexity of the controlling mechanism is mainly due to the many combinations which can be achieved by different connections between layers.

The function of layer II elements is to process symbolic data belonging to the same object, therefore the function of the communication network is to collect the symbolic data from different layer I elements and pass them to a layer II element. The many to many mapping and the many to one to many mapping are both potential candidates for the network because both demonstrate the collecting effect. To derive the most suitable communication network, factors of efficiency, fault tolerance and expandability will be compared.

7.3.2.1 Efficiency Analysis

In order to compare the efficiency of the two networks, a quantitative analysis will be carried out but prior to that, a description of the operations performed by the networks during data transfer from layer I to layer II is given.

If the layer I and layer II arrays are connected by the many to many network, each layer II element will be connected to 16 consecutive layer I elements. If the term object can refer to a group of connected pixels having similar properties, such as intensity level, it can be assumed that data processed by consecutive layer I PEs is likely to come from the same object. Since each PE in layer II is provided for processing symbolic data derived from the same object, it is connected to 16 consecutive PEs in layer I. In order to transfer data from layer I to layer II, each layer I element will incorporate an output buffer from which layer II elements will access the data, as shown in Fig. 7.4. It is assumed that the symbolic data is arranged in a 2-D matrix having the dimension of 256*N. The value of N is dependent on the quantity of symbolic data extracted. The matrix is divided into N rows and symbolic data is transferred from layer I to layer II row by row through layer I's output buffers. Because not all data in a row is valid, some of the buffers may be empty. Instead of reading all 16 output buffers one by one, a layer II element will only read data from the non-empty buffers. This is achieved by a decoding network, described in Chapter 8. The next row of data is not available until all layer II elements have finished reading the data from the output buffers connected to them. The process continues when layer I outputs another row of data and can be summarised in the following steps:

1. Layer I elements load a row of data into the output buffers.

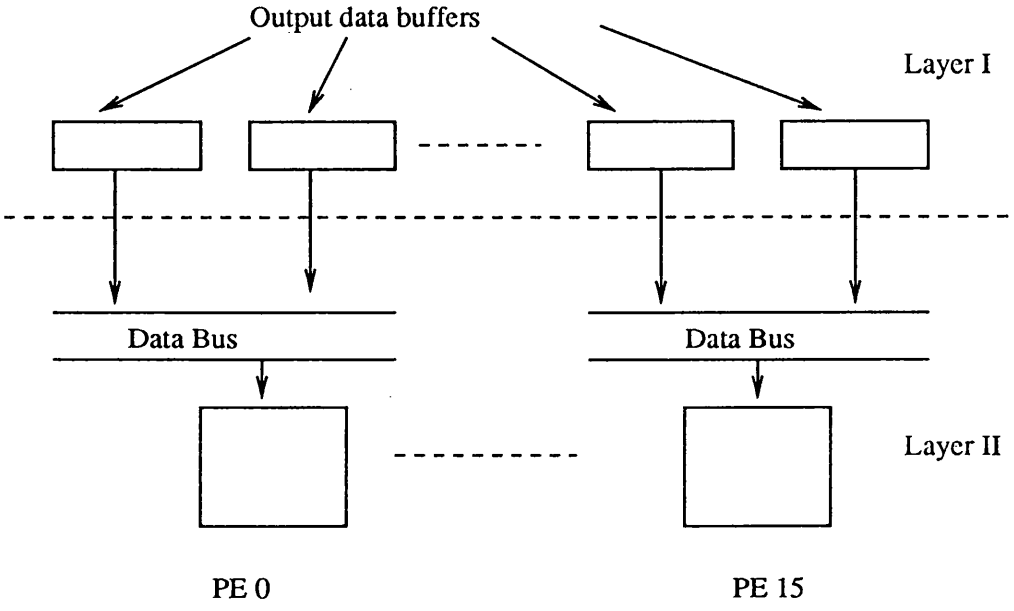
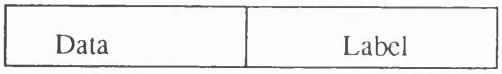


Figure 7.4 An overview of the layer I and layer II communication network

2. Layer II elements examine the output buffers and locate the non-empty buffers via a decoding network.
3. Layer II elements read data from the non-empty buffers only.
4. A layer II element waits if other elements have not completed reading their data.
5. Repeat from (1) if there are data to transmit.

In the many to one to many network, the process can be divided into two stages. In the first, data is read from layer I to the dealer, and in the second passed to layer II's elements. Operations performed in the first stage are very similar to those described in the many to many network: the layer I elements again load a vector of data to the output buffers and similarly a decoding network is provided for locating the non-empty buffers. The dealer then reads data stored in those buffers and, according to its value, selects a layer II element to pass it to.

Data passed from layer I to layer II is regarded as symbols. Each symbol is a collection of data and it is assumed that a label is included. Data is encoded by a label, as shown in Fig. 7.5.



A symbolic data

Fig. 7.5 Format of the symbolic data

Symbols belonging to the same object will have the same label value, which can then be used to distinguish symbolic data coming from different objects and also to select the proper layer II element by the dealer. A decoding procedure is required when the dealer transfers data to a layer II element and can be achieved by either software or hardware. The software solution might be a lookup table algorithm, using the label value as an index to address a lookup table, which stores values representing layer II elements. The hardware solution could be a demultiplexer. Since there are only 16 elements in layer II, 4 bits from the label are sufficient for the decoding purpose and a 4-line-to-16-line decoder [66] can be used. The hardware solution is adopted because it can speed up the process and details of the circuitry are included in Chapter 8. The complete data transfer procedure is divided into the following stages:

1. Layer I elements load a vector of data into the output buffers.

2. The dealer examines the output buffers and locates the non-empty buffers.
3. The dealer reads data from the non-empty buffers.
4. The dealer transfers the data to the second layer elements and repeats (3) until all the data are read.
5. The above is repeated as long as there are data to send.

After the above description of the functions of the networks, an analysis of the efficiency of the networks can proceed. In order to carry out the analysis the following assumptions are made:

1. 16 objects are extracted from the input image;
2. Each object comprises n pieces of symbolic data;
3. Transfer data from layer I PEs to layer II PEs in the many to many network and from a layer I PE to the dealer in the many to one to many network take D clock cycles;
4. It takes d clock cycles to transfer data from the dealer to a layer II PE.

The overall data passing process can be divided into two stages, which are:

1. Transferring data from layer I to layer II; and
2. Exchange of data between PEs in layer II.

Using the above assumptions, the time required to perform the data transfer can be evaluated in terms of the unit of clock cycle employed by the system.

Passing Data from Layer I to Layer II

The many to many network is examined first. As described previously, each element in layer II communicates with 16 layer I elements via their output buffers and will only access an output buffer if it is not empty. A special decoding circuit is devised to select the non-empty buffers automatically and the time required by the process is negligible comparing with the total time taken to transfer data between the layers. From this the time taken to transfer a vector of data from layer I to layer II can be expressed as $D^* \max\{X\}$ clock cycles. The term X denotes the number of non-empty buffers in every 16 output buffers. The 16 layer II elements each has 16 buffers to read operating in parallel. Since the next row of data is not available until all layer II elements have completed reading the data, the time required to complete the reading of a row of data equals the maximum amount of data read by a layer II element. The maximum value of max{X} is 16 so the total time required to transfer all data in the matrix can be expressed as

$$D^{*}\sum_{i=0}^{N-1}\max\{X_i\} \tag{7.1}$$

N is the number of rows of the matrix accommodating the symbolic data (as described in previous paragraph). Where X_i denotes the amount of data read by a PE at the ith row of the matrix. In an ideal situation where every layer II element performs the same number of reads at each row of data, then equation (7.1) gives the value n*D (the minimum time). The maximum value of equation (7.1), n*D*16, occurs when all the data are confined to a single PE in layer II. The time taken to transfer data from layer I to layer II in the many to many network therefore varies from n*D to n*D*16 clock cycles.

For the many to one to many network, the process of transferring data from layer I to layer II is divided into two steps. The first step involves passing data from layer I to the dealer, and the second passing data from the dealer to layer II's elements. Passing data from layer I to the dealer is a similar process to that described in the many to many network. Once again only the non-empty buffers of the layer I elements will be accessed and it is assumed that the time required to locate them is negligible. The maximum number of non-empty buffers being 256 and the time taken to transfer all the symbolic data from layer I to the dealer is equal to $D^{*}\sum_{i=0}^{N-1}X_i$ clock cycles, where X_i represents the number of non-empty buffers appearing at row i and $\sum_{i=0}^{N-1}X_i$ equals the total amount of data available, i.e. n*16. It therefore takes D*n*16 clock cycles to transfer all the data from layer I to the dealer.

Once a datum arrives at the dealer, the dealer selects a layer II element and directs the datum to it. It is assumed that it takes d clock cycles to achieve that and for a total of 16*n data, it will take d*n*16 cycles to pass all the data from the dealer to layer II. The total time required to transfer all the symbolic data from layer I to layer II in the many to one to many network is 16*n*(D+d) clock cycles.

Inter-processor Communication

The data received by a layer II element may not belong to the same object and it must be transferred to the proper element before intermediate-level operations in layer II can proceed. In an ideal case, where every element received the same amount of data belonging to the same object, no inter-processor communication would be required but in a general situation, inter-processor communication is very likely to take place and the time taken to carry out the operations can be estimated. The data passing mechanism employed in the proposed system is first described.

It is assumed that two 16-bit data channels, formed by a set of data buffers and resembling the D-chains of the CLIP7A system, are provided for intra-layer communication in layer II. Data moves along the channel by shifting from buffer to buffer and data in both channels makes the same number of moves but in opposite direction. There are two data buffers labelled LHS (Left-hand-side) buffer and RHS (Right-hand-side) buffer in each layer II PE. The 16-bit data channel formed by connecting all the LHS buffers carries data from a PE to its left-hand side neighbours, whilst data travels in the opposite direction via the channel formed by the RHS buffers. There is a connection between the buffers at the ends of a channel so that data can cycle through a channel. By loading the data into the proper channel, the data can reach its destination in a maximum of eight shifts. Passing data from one element to another will require the following operations:

1. Each element examines the data it received and identifies those data to be transmitted to its neighbours. Data belonging to the neighbours is divided into two groups: left-hand neighbour and right-hand neighbour
2. Each element loads its LHS buffer with data from the left-hand neighbour group, with the RHS buffer loaded with data from the right-hand neighbour group
3. Data loaded to the buffers are shifted along both data channels
4. Each element reads the data received in its buffer and decides whether to store the data. If the data is stored then the corresponding data buffer is cleared
5. The channel controller examines the data channels; if there are data in either channel then steps from (3) onwards are repeated
6. If there are still data to be transferred from any PE then the process is repeated from step (1)

Based on the above data passing mechanism, the maximum number of shifts which are needed is governed by the maximum distance travelled by a piece of data. The inter-processor communication time can be represented by equation (7.2).

$$C_b*M+C_r*\sum_{i=0}^{K-1}\max\{T_i\} \tag{7.2}$$

M represents the maximum number of data, transmitted from layer I, received by a layer II element and it can be up to n*16. C_b represents the time taken by a layer II element to examine an input data coming from layer I. The term C_b*M corresponds to the time taken by layer II elements to examine the input data before deciding to pass them to their siblings. Since all layer II elements examine their data in parallel and

elements will not begin transferring their data until all elements finish examining it, the time taken for this procedure is determined by the maximum amount of data received (M) by a layer II element. C_r represents the time required to examine the input data coming from a PE's siblings. The term $C_r \sum_{i=0}^{K-1} \max\{T_i\}$ represents the time required to exchange all the data between the layer II elements. As described above, an inter-element communication cycle completes when all the data in both channels have been read. The time taken for such a cycle is determined by the maximum distance travelled by a piece of data in either channels, i.e. max{T}. The total number of communication cycles required to exchange all the data between the layer II elements is K. The maximum value of K is 8*n, when all data transmitted from layer I are collected by a single layer II element. The term T_i is the distance travelled by a datum at the ith communication cycle, the maximum value of T_i is 8.

The inter-element communication time will vary from n*C_b (for the ideal situation) to (16)*n*C_b + 48*n*C_r (when the value of K is maximum). When K is 8*n, it implies that all the data are collected by a single layer II element. In order to redistribute the data, 2*n pieces of data have to be travelled the distance of 1, 2, ..., 7 accordingly and n pieces of data travel the distance of 8. The term 48*n is obtained after considering all the possible combinations that can be derived in order to redistribute all the data.

For the many to one to many configuration, because the dealer also distributes the data to the proper PE, so inter-processor communication at this stage is not required. So the total communication time required by the two networks is:

$$D*\sum_{i=0}^{N-1} \max\{X_i\} + C_r*\sum_{i=0}^{K-1} \max\{T_i\} + C_b*M \qquad (7.3)$$

for the many to many mapping; and

$$16*n*(D+d) \qquad (7.4)$$

for the many to one to many mapping.

In order to continue the analysis, the magnitude of D, d, C_r and C_b will be estimated. The Motorola MC68000 microprocessor is proposed as the computing units for the system (explained in Chapter 8) so the values for D, d, C_r, C_b are estimated using information relevant to the microprocessor [67]. The symbol D represents the time taken to read data from an output buffer of a layer I element and store it into the local memory of a layer II PE or some temporary storage. This requires about 20 clock cycles which account for moving the data from the local memory of a layer I element

to its output buffer and loading the data to a temporary storage. The term d represents the time taken to pass data from the dealer to a layer II element. When considering the implementation of the many to one to many network, data being read from an output buffer will not be stored in the dealer's memory, so the term d is regarded as a decoding delay which is required to select the layer II element. As previously stated, the hardware solution will be applied and the estimated value of d is 5 clock cycles. For the many to one to many network, by substituting the values of D and d into equation (7.4) the time taken to pass data from layer I to layer II will take 16*25*n clock cycles, where n is the number of symbolic data representing an object.

The symbol C_b and C_r represent the time required to complete the two phases during inter-layer communication of the many to many network. The first phase is called broadcasting, which represents the operations performed by a layer II element to examine the data received and determines the data to be transferred to its siblings. It takes C_b clock cycles. The second phase is receiving: each layer II element examines the data coming from its neighbours, taking C_r clock cycles. In order to estimate the values of C_r and C_b, pseudo-codes which represent the operations to be performed at each phase are devised.

Broadcasting

```
Begin{
    read datum
    check label value
    if header not equal to PE's number then
        if data belong to left-hand neighbours then
            load datum to LHS group storage
        end if else
            load datum to RHS group storage
    end if
    read next datum
    }
```

Receiving

```
Begin{
    read datum from LHS (or RHS) buffer
    check label value
    if header equal to PE's number then
        store datum
        clear LHS (or RHS) buffer
    end if
    }
```

Using the Motorola 68000 microprocessor as the model processor, each set of codes takes about 50 clock cycles to execute. Since each layer II element has to examine two data buffers (LHS and RHS), C_r is 100 cycles. Substituting the estimated figures for D, C_r and C_b into equation (7.3), the following equation is obtained:

$$20\sum_{i=0}^{N-1}\max\{X_i\}+50M+100\sum_{i=0}^{K-1}\max\{T_i\} \qquad (7.5)$$

There are still variables which cannot be predicted at this stage. The term $\sum_{i=0}^{N-1}\max\{X_i\}$ represents the total number of read operations performed by the layer II elements when layer I transmits the symbolic data matrix. The value M is the maximum number of data received by a layer II element. $\sum_{i=0}^{K-1}\max\{T_i\}$ is the total number of read operations performed in order to exchange data between the layer II elements. Using the benchmark images as an example, each rectangle is represented by its intensity, depth and X, Y coordinates of the corners, the maximum value of n is 10 (when all four corners of a rectangle are located). If n equals 10 then the many to one to many mapping will take 4000 clock cycles to complete the data passing. For the many to many network, the operation will depend on the values of $\sum_{i=0}^{N-1}\max\{X_i\}$, M and $\sum_{i=0}^{K-1}\max\{T_i\}$. In order to simplify the mathematical notation the terms $\sum_{i=0}^{N-1}\max\{X_i\}$ and $\sum_{i=0}^{K-1}\max\{T_i\}$ are represented by A and B respectively. The values of A, B and M under the above conditions will have the following values:

A varies from 10 to 160;

B varies from 0 to 480; and

M varies from 10 to 160.

The minimum time required to transfer all the data from layer I elements to proper

layer II elements is 700 clock cycles, when the values of A, B and M are 10, 0 and 10 respectively. If the values of A, B and M are at their maximum then the maximum time required (59200 clock cycles) is obtained. The time required for data passing in the many to many network varies from 700 to 59200 clock cycles and is related to the distribution of objects in an image. Three examples which demonstrate how the distribution of objects affecting the data passing mechanism are given. If objects within an image are evenly distributed then the values of A, B and M will be rather small, for example 20, 8 and 13. This gives an execution time of the order of 1850 clock cycles, about half the value required by the many to one to many method. However, if data about two objects are collected by a single layer II element then the value of M will be in the order of 20. If the data have to be transferred to an element four units away, then the value of B will be 40. The minimum value of A in this situation is 20. According to these data, the estimated communication time is 5400 clock cycles, which is about 35% longer than the many to one to many mapping . The values of B and M will increase if the distribution of objects in the input image is not even. If the values of A, B and M are 30, 50 and 30 then the time required becomes 7100 clock cycles which is about 78% longer than the many to one to many mapping. Because the magnitudes of the terms C_r and C_b of the many to many network are much greater than D, so when intra-layer communication is frequent then the effectiveness of the network is reduced.

From the above discussion, it is difficult to decide which mapping is more efficient. The performance of the many to many mapping is governed by the distribution of objects in an image, whilst the loss of performance with the many to one to many mapping is directly proportional to the amount of data required to be transferred. Since in addition to the efficiency factor, fault tolerance and expandability must also be considered; a decision will be made after those factors are examined.

7.3.2.2 Fault Tolerance

The major requirement of fault tolerance is to maintain the computing power provided by each layer and the connectivity both between layers and within layer. As described in Chapter 3, Section 3.1, a linear array is relatively easy to achieve fault tolerance because of its one dimensional structure. Maintaining the connection within layer is rather simple, as was explained in Chapter 3 Section 3.1. However, retaining the inter-layer communication network can be rather difficult and both the many to many mapping and the many to one to many mapping will be studied. In each mapping two situations are considered. First, faulty elements arise in layer I and second, in

layer II. It is assumed that spare elements are available at the end of both arrays so that computing power in each layer can be retained. The effect on the connection network when bypassing a faulty element and activating a spare element will be considered.

Many to Many Mapping

When a faulty element appears in layer I, two communication links are affected. The first one is the communication link between the faulty element and an element in layer II. The second link determines how the replacing element is connected to the appropriate element in layer II. The most straightforward solution to this is to connect the replacing element to the layer II element originally connected to the faulty element, as shown in Fig. 7.6. In order to achieve this, there must be a switch, as shown in Fig. 7.6, which enables the establishment of a communication link between the spare element and the layer II element. A faulty element can appear in any position in a layer and a spare element must be able to replace any layer I element. In order to achieve this, a switch must be provided at each communication link. The number of entries to a switch should be proportional to the number of spare elements available in the layer. The presence of the switches will increase the cost of the system, additionally, the control requirements for the switches also increase the complexity of the system. A simpler alternative is to connect the spare element to the last (16th) element in layer II, as shown in Fig. 7.7. In doing so, the switches required in the above arrangement are avoided. The 16th element in layer II will treat the spare element as a normal PE. However, if this 16th element becomes faulty then the structure proposed cannot apply.

The situation is more complex if a faulty element appears in layer II. Each layer II element is connected to 16 layer I elements, so at least 16 communication links are involved when a layer II element fails. Similar to the situation in layer I, both the communication links connected to the faulty element and the replacing (or spare) element have to be considered. The 16 layer I elements connected to the faulty element must re-establish new communication channels with other layer II element(s). One solution is to connect them to the spare element, as shown in Fig. 7.8. In order to achieve this, switches which control the data paths between elements in layer I and layer II must be provided. Alternatively, the layer I elements can transfer their data to the faulty element's nearest neighbours, as shown in Fig. 7.9, whilst the spare element receives data via other layer II elements. Again, re-connecting the layer I elements requires switches to establish the new data paths. There is, however, one solution

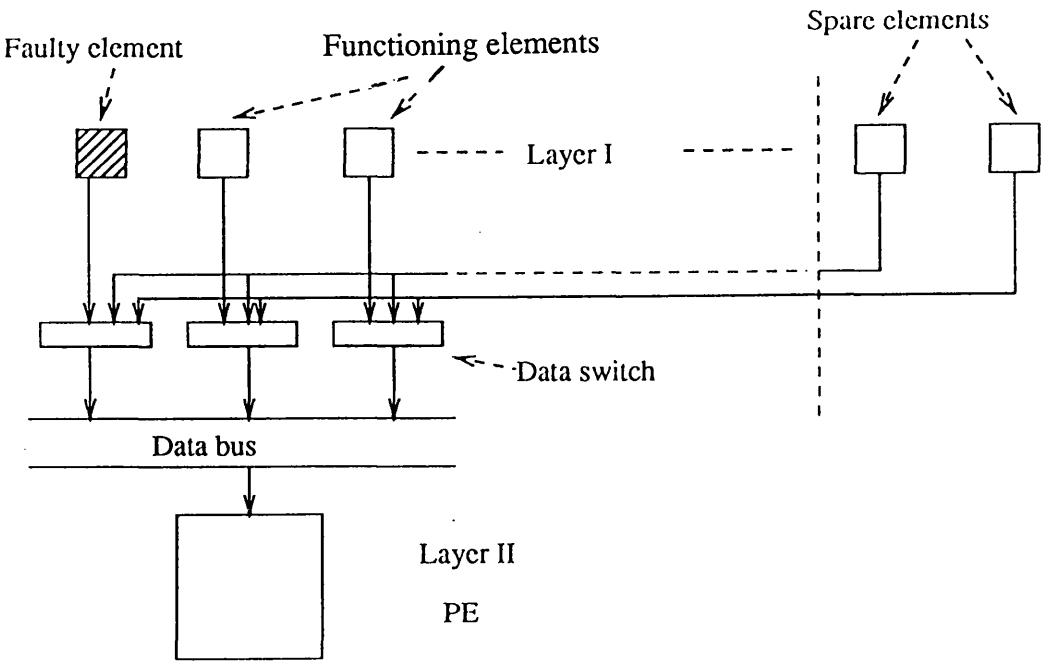


Figure 7.6 A mechanism to achieve fault tolerance in layer I for the many to many network

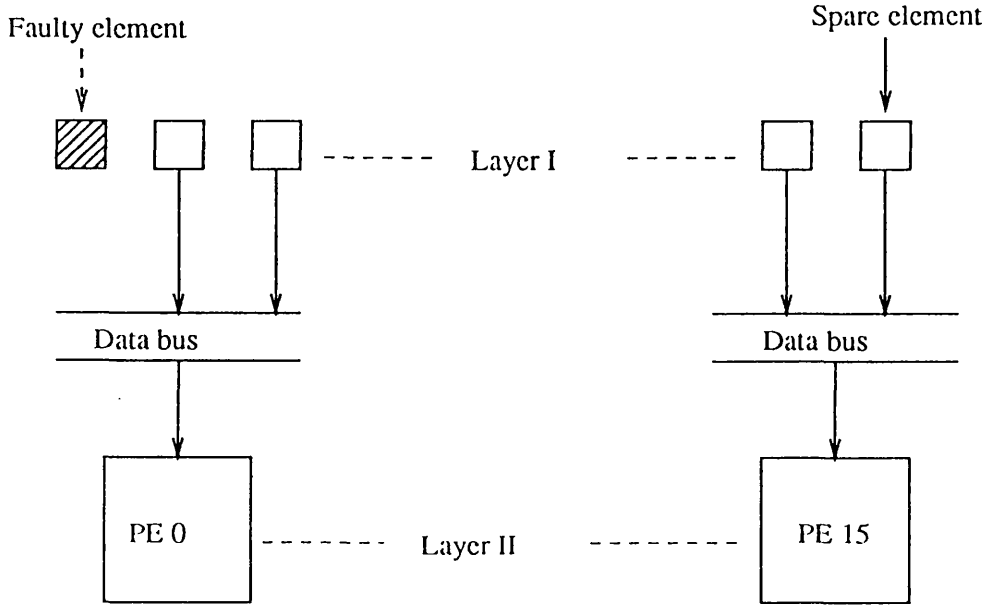


Figure 7.7 An alternative method to achieve fault tolerance in layer I for the many to many network

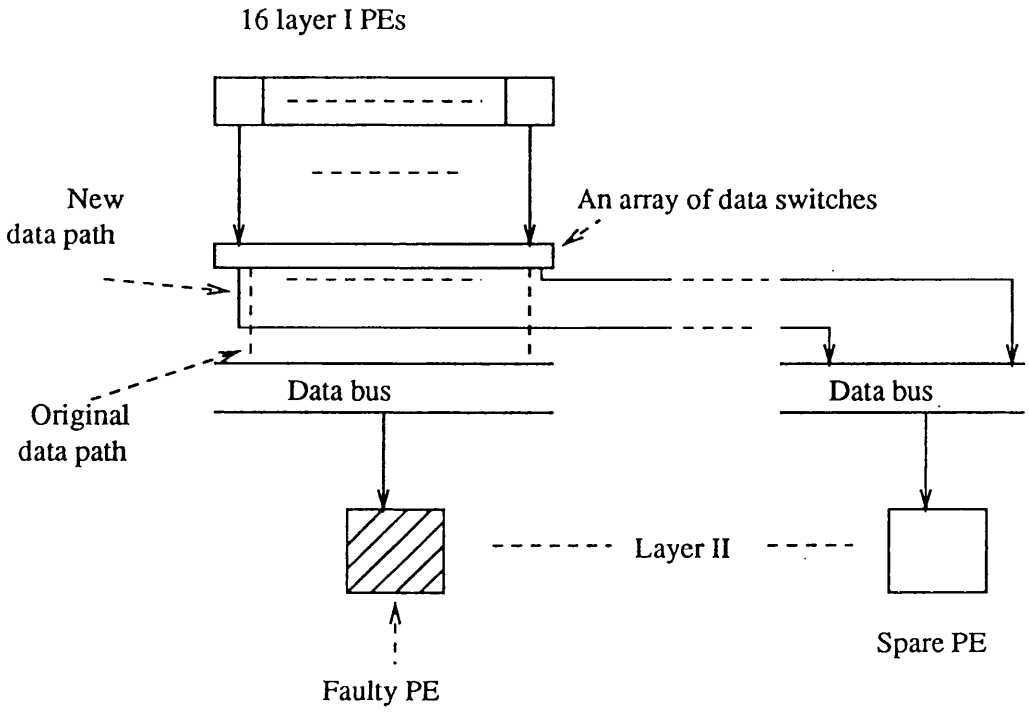


Figure 7.8 A mechanism to achieve fault tolerance in layer II for the many to many network

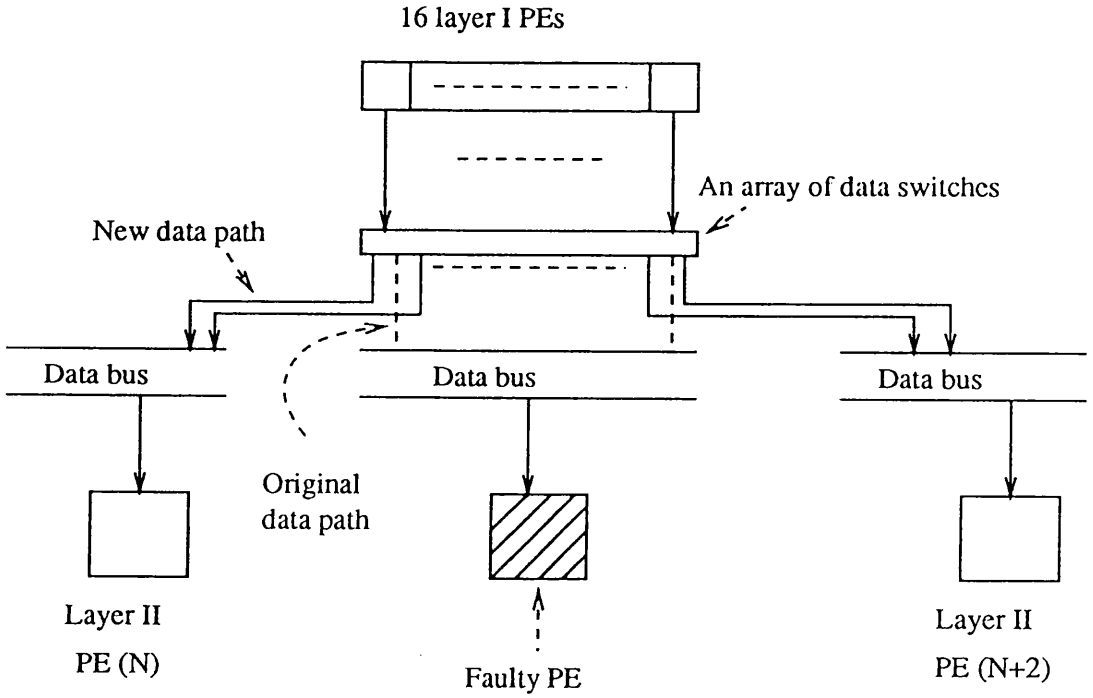


Figure 7.9 Re-directing data paths from a faulty layer II PE to its nearest neighbours

which does not require the switching mechanism to re-direct the data paths. If the 16 layer I elements and the layer II element communicating with them are treated as a single unit then, when a layer II element becomes faulty, the whole unit can be bypassed and replaced by a spare unit, as shown in Fig. 7.10. This solution is relatively costly but can be achieved by a simpler controlling mechanism, because no switching of inter-layer communication paths is required.

Many to One to Many Mapping

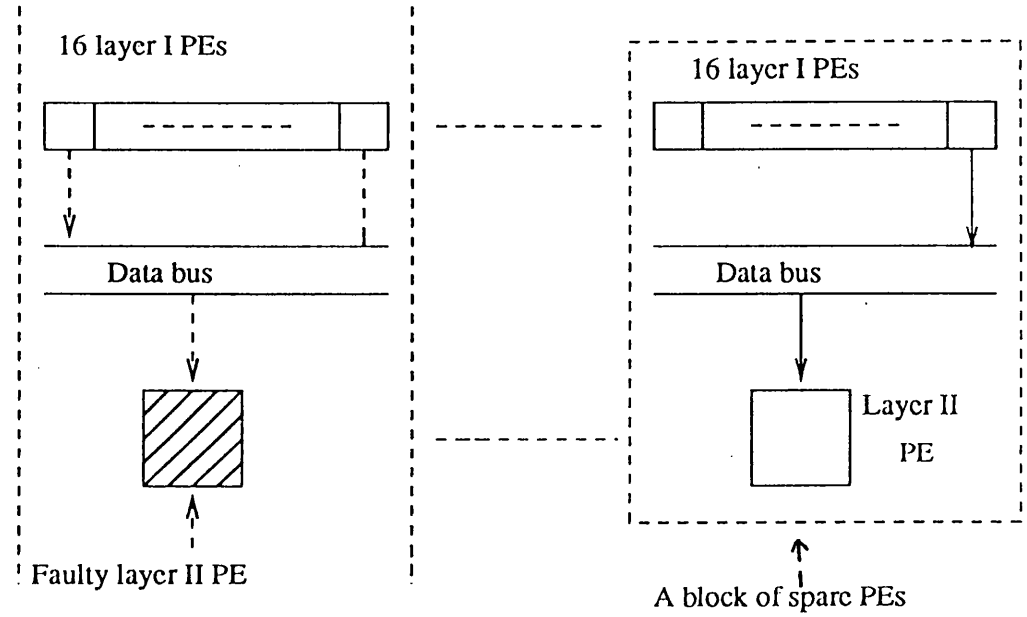
In the many to one to many mapping, fault tolerance is easier to achieve because layer I PEs or layer II PEs are connected to a single element (the dealer). Bypassing or replacing an element in either layers will only involve a single communication link. Moreover, elements in both layers are connected to the dealer, so re-directing the data path from a faulty element is not necessary. When a faulty element appears in layer I then a spare element can be engaged by connecting it to the dealer, as shown in Fig. 7.11. Similarly, spare element in layer II can replace the faulty element by simply making a connection with the dealer. In both cases, existing connection links are not affected.

At the moment, how to keep the dealer fault proof is not discussed. To implement the network, the functions of the dealer are performed by the array's controller, explained in Chapter 8. The design of the controller is not seriously influenced by the functions of the dealer. The effort required to achieve fault tolerance in the controller in this connection network will be the same as those in the many to many network.

In such a case, the conclusion that fault tolerance can be achieved easily by the many to one to many network is still valid.

7.3.2.3 Expandability

Expandability refers to the possibility of adding extra elements to layer II in order to improve the parallelism, as discussed in Section 7.2. The major concern is how communication between the extra element(s) and layer I's elements can be achieved. In the many to many mapping, because each element is physically connected to a group of layer I elements, so adding extra elements to layer II will affect the original connections. If each layer II PE has to have the same number of connections with the layer I PEs then the complete connection network has to be re-defined. In the present situation each layer II PE is connected to 16 layer I PEs. If 4 elements are added to layer II then each element will be connected to either 12 or 13 layer I elements. Alternatively, the additional PEs need not connect to any layer I PE and need



Bypassing a block of PEs

Figure 7.10 Achieving fault tolerance by providing a block of spare PEs

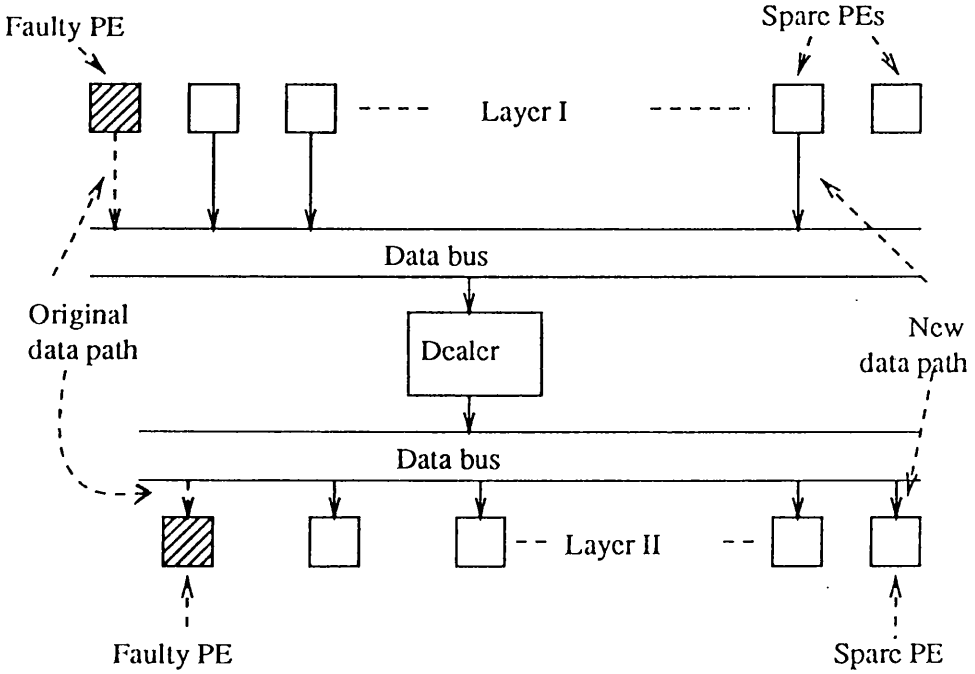


Figure 7.11 Connecting spare elements in the many to one to many network

only receive data transferred from their peers. In such a case, the inter-layer connectivity can be retained, but the performance of the system may be degraded because of the increasing intra-layer communication, as discussed in Section 7.3.2.1.

In the many to one to many mapping, adding extra elements to layer II can be achieved easily. This is because all layer II PEs are connected to the dealer only: adding extra elements will not affect other communication links which already exist, as shown in Fig. 7.12. The extra elements will establish connection links with the dealer.

7.3.2.4 Conclusion

From the above discussion, it is concluded that the many to one to many mapping is more flexible and robust than the many to many mapping. The many to many mapping, however, is more efficient in some cases.

In the above example, it was assumed that the number of objects in the input image was equal to the number of layer II elements. As discussed in Section 7.2, an element in layer II may be assigned to operate on more than one object at some situations. In those cases, because of the intra-layer communication, the effectiveness of the many to many network will be degraded. Moreover, it is difficult to maintain fault tolerance and still permit expansion in layer II. It is concluded that the many to one to many mapping is more suitable for the inter-layer communication network between layer I and layer II. In Chapter 8, the implementation of the network will be described.

7.3.3 The Layer II and Layer III Communication Network

As discussed in the previous sections, the design of an inter-layer communication network has to be based on the network's efficiency, expandability and its capability to achieve fault tolerance. The efficiency of the network is determined by how well the structure of the problem is mapped onto by the network.

Processing elements in layer II operate on data which belong to a single object and produce a description of such an object. Elements in layer III will collect all the information available in layer II in order to create an overall description of the input image. The layer III elements will perform model matching or database searching. Each element will embody a different database file or part of a huge data file, so either model matching or database searching can be performed in parallel, with each layer III element searching or matching the input data with its own data file. Since each layer III element requires all the information provided by layer II elements, a connection network which enables layer II elements to broadcast data to all layer III elements

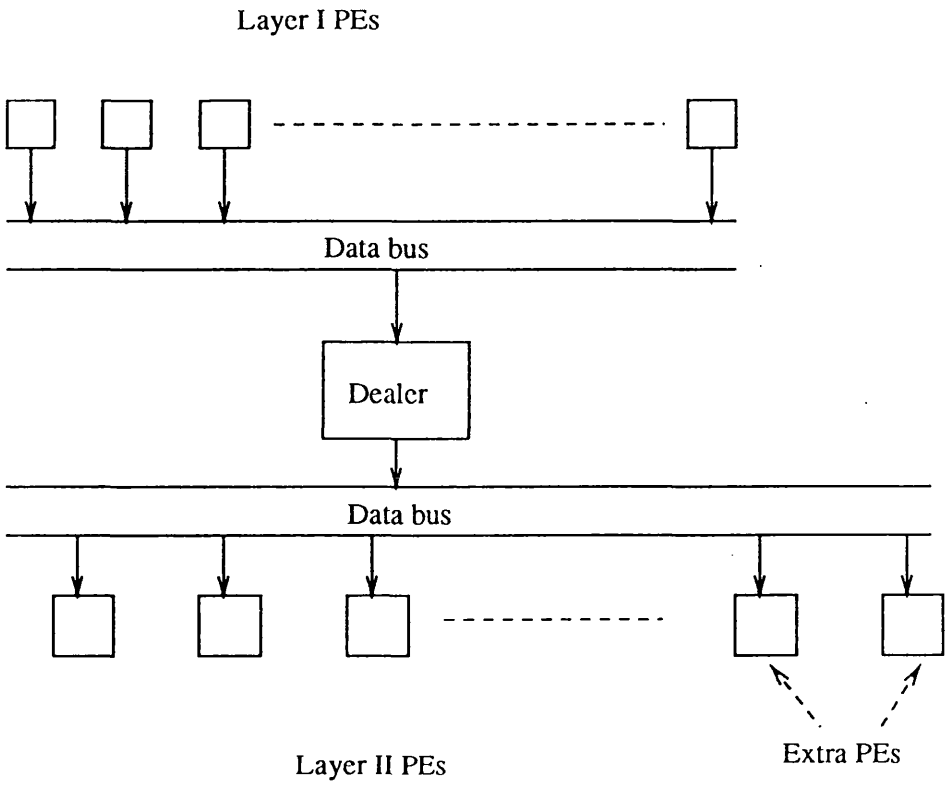


Figure 7.12 Adding extra PEs to layer II in the many to one to many network

is appropriate.

A single communication bus structure is proposed. Each element in layer II operates on a unique set of data, so it is possible to assume that operations performed by them may take different periods of time to complete. It is, therefore, possible and economical for the elements to share a single bus. By providing a single communication bus, this can minimise the cost of the system and simplify the controlling mechanism. The proposed structure, as shown in Fig. 7.13, embodies a single 16-bit bus which is connected to all the elements in both layer II and layer III. Because there is only a single bus, so only one element in layer II can occupy the bus at each communication cycle. A communication cycle refers to the operations performed when a piece of data is passed from a layer II element to a layer III element. When a layer II element occupies the bus, it will output data to the communication bus and all the elements in layer III will receive the data. The major design criterion of the communication structure is to achieve bus arbitration. The controlling mechanism must cater for the following situations:

1. Only one layer II element can occupy the communication bus
2. When the bus is already occupied then an element ready to send data has to wait
3. When several elements are preparing to broadcast their data at the same instant then only the PE with the highest priority is permitted to send its data.

As previously stated, the computing unit proposed for layer II is the MC68000 microprocessor. It is assumed that a temporary storage is provided in a layer III PE and data available in the data bus can be loaded into the storage without the interaction of the layer III elements. According to the information obtained for the MC68000 microprocessor, to move a piece of data from a layer II element to every layer III element will take about 16 clock cycles, once the communication bus is granted. The time taken to transfer data from layer II to layer III is directly proportional to the quantity of data available in layer II. Since bus arbitration and data passing can be performed in parallel, moreover, once an element is granted the bus no further arbitration process is required. Therefore, we may assume that the arbitration process will only take a fraction of the data passing time. The benchmark image is again used as an example, for each rectangle 8 parameters are derived including the length of both axes, the coordinates of the centre, orientation, intensity, depth and the rectangle number. A total of 128 words of data will be transmitted from layer II and assuming that the arbitration process takes 10% of the data passing time then the total time required to transfer all the data is 2260 clock cycles.

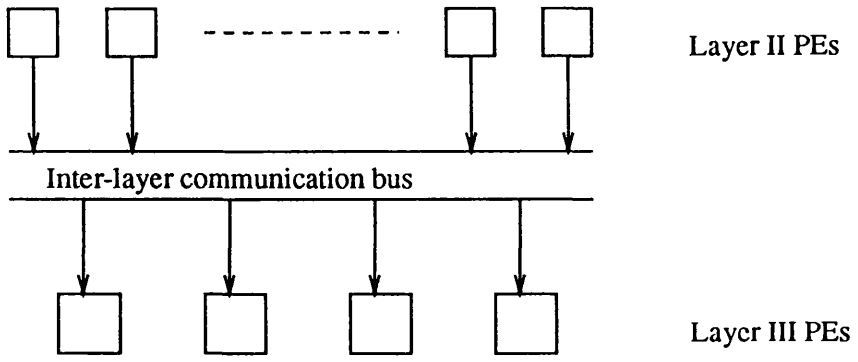


Figure 7.13 Block diagram of the inter-layer communication network between layer II and layer III

The bus structure can achieve expandability and fault tolerance rather easily. This is because there is no direct connection between elements in the two layers. Adding extra elements or bypassing faulty elements will only affect the communication links connecting those elements to the data bus. Elements in both layers are connected to the communication bus for either input or output and adding extra elements in either layer just requires connecting the new element to the bus, as shown in Fig. 7.14. When a faulty element is found in a layer, it will be bypassed and a spare element will be activated by the layer's controller.

Since the network controller is responsible for scheduling the data passing between elements in both layers, it must be notified of the presence of any extra elements or faulty elements. The network controller must be programmable in order to support fault tolerance and expandability and further discussion is given in Chapter 8.

7.4 Conclusion

In this Chapter, the overall structure of the proposed three layer system has been described. The number of processing elements employed by each layer, beginning from the first layer, is 256, 16 and 4. To improve a network's efficiency, expandability and capability to achieve fault tolerance, suitable inter-layer connection networks between layer I/II and layer II/III have been chosen. The connection network between layer I and II is called many to one to many because of its collecting and distributing property. Data transfer between elements in layer II and III is achieved by a single communication bus. Estimates given in Section 7.3 suggest that the total time required for communication between layers is about 6260 clock cycles for the benchmark images (4000 cycles are required when layer I transferring data to layer II, whilst it takes about 2260 cycles to move data from layer II to III). If the system uses a 10MHz clock then the communication time is equal to 0.63ms. Comparing with the processing time (40ms) of a frame of an image in a video-rate application (the most time demanding situation) inter-layer communication is about 2% of the processing time. The estimated time is low compared with the video frame time. This suggests that the design is effective even when the number of data to be transmitted between layers increases. In the next Chapter, the physical implementation of the networks and other system parameters will be discussed.

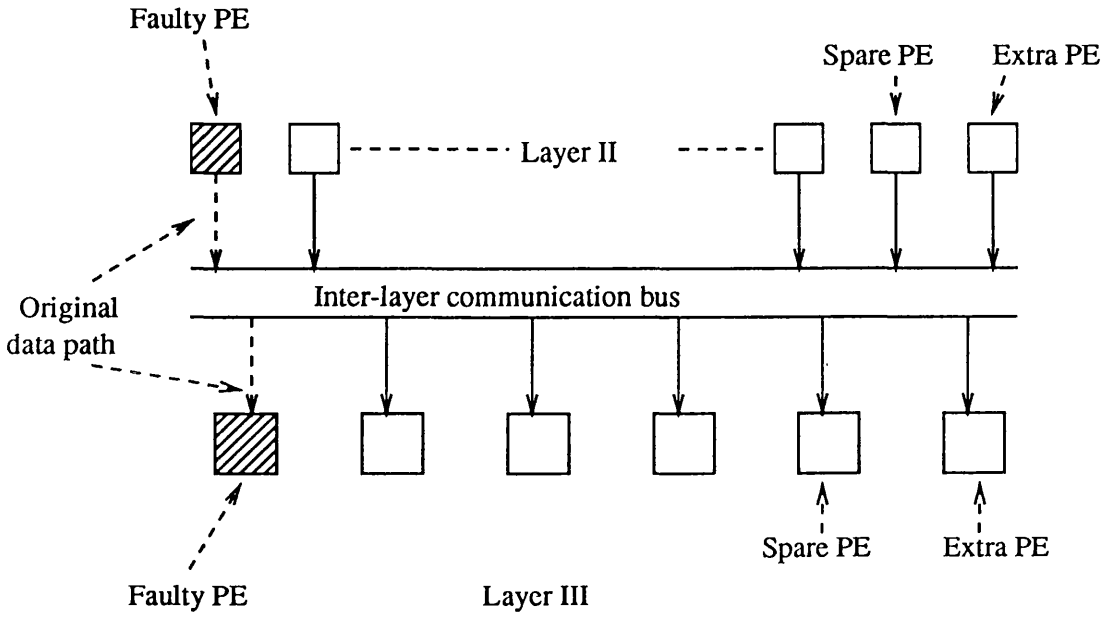


Figure 7.14 Expanding and achieving fault tolerance in the proposed layer II to layer III communication network

Chapter 8
The Design of Some System Components

8.1 Introduction

In Chapter 7 the general structure of the proposed multi-layer system, incorporating three units, was introduced. Of these units, the functions of the host computer and the video I/O unit were briefly described in the previous chapter, which also included a detailed discussion of the inter-layer communication networks to be implemented in the multi-layer unit. The multi-layer unit remains to be considered. It can be divided into eight components, which consist of three linear arrays, two inter-layer connection networks and three controllers, as shown in Fig. 7.2. In this chapter the structures of these components are introduced and a brief discussion of the system software is included.

8.2 The Design of the Linear Arrays

The design of the linear arrays is based on results obtained from the investigation described in chapters 5 and 6. Computing power provided at each layer and inter-element communication are two major criteria to be considered in the design, but factors of cost and compatibility between layers are also important. The first stage of the design process is to select the computing devices to be employed in each layer.

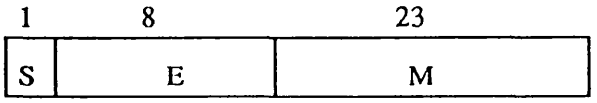
8.2.1 The Computing Requirements of the Multi-layer Unit

The choice of computing devices to be employed by each layer is determined by three factors: processing requirements, cost and compatibility. Processing requirements for the linear arrays are considered first and are related to the different levels of processing involved.

The layer I array performs low-level operations on the input image: the survey presented in Chapter 3 Section 3.3 showed that various computing units, ranging from single bit to 16-bit devices, could be used. Each pixel in an input image carries 8-bit information (see Chapter 7 Section 7.1) and the computing unit should, therefore, be able to perform 8-bit computations effectively. In many low-level operations such as convolution, intermediate results requiring more than 8-bit accuracy are generated and the computing unit must be capable of performing these computations. These requirements imply that the computing unit could be either a 8-bit or 16-bit device. Local addressing, as described in Chapter 5 and Chapter 6, can improve the efficiency of a linear array in various operations and should be included in each layer.

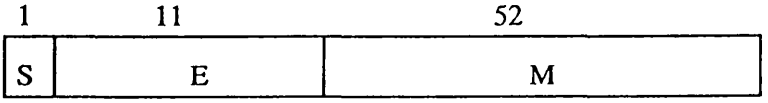
The survey presented in Chapter 6 Section 6.1 and the investigation conducted in Section 6.3 demonstrated that the computing requirements for layers II and III are very similar and they are consequently discussed together. The first requirements are for high-precision or floating-point computation and special mathematical functions. The need for floating-point processing is prominent in the original benchmark problem which includes an image presented in the form of floating-point numbers. In other problems, such as the human face profile, floating-point representation is utilised in the evaluation of the mean and standard deviation from a set of parameters. Mathematical functions, for example square root and trigonometric functions, are commonly used. The evaluation of the distance between two points in an image from their X and Y coordinates, for example, requires the square root function. Results obtained from these computations are also often expressed in floating-point format.

There are two commonly adopted floating-point representations [68]: 32-bit and 64-bit. Each floating-point number embodies three pieces of information including sign, exponent and mantissa, as shown in Fig. 8.1. There is one sign bit in both representations. The 32-bit format provides 8 bits for the exponent and 23 bits for the mantissa, the 64-bit format provides a 11-bit exponent and 52-bit mantissa. Obviously, the 64-bit format enables more accurate computation to be performed but is more time consuming, assuming that only a 32-bit or 16-bit computing device is available. The accuracy of the computation also depends on the resolution of an image, which can be expressed in terms of the number of pixels per centimetre or metre, depending on the actual size of an object. If the resolution of an image is low then using the 64-bit format will not provide significant improvement in the results. Because of the size of the image being used and the problems that we are interested in, the resolution of the input image will be relatively low and the 32-bit floating-point format should provide sufficient accuracy for the computation. Since the 32-bit format will be adopted, a 32-bit computing device seems appropriate, but a 16-bit device is proposed based on the grounds of cost. As an example, the cost of a 16-bit microprocessor (the Motorola 68000) is about £7 whilst a 32-bit microprocessor (the Motorola 68020) is about £100. The 32-bit device is about 14 times more expensive. The 16-bit device to be selected, however, should be flexible so that 32-bit computation can be carried out easily. Floating-point arithmetic and special mathematical functions will be implemented in software in the current design, i.e. subroutines for floating-point arithmetic will be provided, and this is also based on the factor of cost-effectiveness. Floating-point processors provide an alternative to the software approach but if only a few functions are being used in a particular problem then a



S -- Sign
E -- Exponent
M -- Mantissa

32-bit floating-point format



64-bit floating-point format

Figure 8.1 Different floating-point formats

hardware solution may not be cost-effective because of the relatively high cost (for example the Intel 8087 floating-point processor is about £80) of those processors. Other criteria that the computing unit should satisfy include programmability and the ability to address a wide range of memory. In layer III, where model matching or database searching is carried out, a more complex algorithm is needed in order to achieve the searching or matching tasks - implying that the processing element should be flexible and easy to programme. Again in layer III, a wide range of memory space is needed to store the database file, or model data. The Chinese character recognition problem, for example, includes of a database file representing 50,000 characters. If each character is represented by 10 bytes of data then the file will occupy about 0.5 Mbytes of memory. Additionally, the complex algorithm may also occupy a substantial amount of memory.

The different computing requirements that the system must cater for have been discussed and the units will be selected from off-the-shelf components for the following reasons:

1. Using off-the-shelf components can reduce both development time and cost of the system;
2. Off-the-shelf components are more reliable;
3. Off-the-shelf components, such as existing microprocessors, always provide a comprehensive programming language that will assist the development of the system software.

There is a huge range of microprocessors, or digital signal processors, which satisfy the computing requirements and, in the following section, the selection is considered.

8.2.2 Choosing the Right Chips

The performance of the system is determined by the computing power of its processing elements. This implies the most powerful chips should be employed in order to maximise the performance. However, in the design of the multi-layer system, cost is a major criterion being considered and the computing device proposed is, therefore, not the ultimate powerful processor. Moreover, the most important feature of the multi-layer system is its layered structure which provides both functional and spatial parallelism. It is, therefore, more important to examine the benefit provided by such a structure (see Chapter 9) than by employing a more powerful computing device.

In layer I, either an 8-bit, or 16-bit, device can be used, whilst layers II and III employ a 16-bit device. Local addressing is one common feature that these devices

must possess and the devices in layers II and III should be easy to programme and capable of addressing a wide range of memory. There is a huge selection of computing devices which embody these properties. The Z80 [69] and 6800 [69], for example, both have a 8-bit CPU that can be used as the layer I computing unit, whilst the Intel 8086 [52] and the transputer T222 [70] are examples of 16-bit processors.

Since a 16-bit device is also suitable for the applications of layer I, one approach to the selection of the computing units would be to employ the same 16-bit device for all three layers. Alternatively, layer I could employ an 8-bit device and layers II and III a 16-bit device. The latter approach is more economical because a 8-bit device is relatively cheaper - as an example, the Z80B microprocessor costs about £2. However, the 8-bit device and the 16-bit device have their own programming languages which might hinder the development of the system software. As discussed in [52], 8-bit microprocessors can be regarded as the first generation general processing units and suffer from other shortcomings such as limited address space (usually 64 Kbytes) and difficulties in programming.

Although employing a 16-bit device in layer I will increase the cost of the system, there are several advantages of such an arrangement.

1. Most 16-bit devices have higher functioning frequencies and can address a wider range of memory;
2. With proper arrangement of data, a 16-bit device could operate on two pixels in a single operation in some functions;
3. If all three layers use the same computing device then both the controlling mechanism and programming methodology for the system can be more consistent; and
4. Using a 16-bit device can improve the robustness of the system. This applies to future applications when pixels might be represented by data with more than 8-bit resolution.

Three 16-bit devices are studied: the Motorola MC68000 [71], the Intel 8086 and the Inmos T222 transputer. Some of their parameters are listed in Table 8.1. The first two are widely used in different personal computer designs and the transputer family is utilised in many parallel systems, such as [72] and [73]. The devices belong to families of microprocessors, which embody devices with different complexity. The Inmos transputer family, for example, includes different 32-bit devices, one of which comes with a floating-point processing unit. The most important feature of these families is their compatibility in software. With the continuously decreasing in price of these

processors, it may be affordable in the future to upgrade the system by utilising more powerful devices, for example 32-bit, with minimal alteration to the existing software.

The Intel device is less attractive because of its relatively high cost and low functioning frequency. The Intel device can address up to 1 Mbytes of external memory but internally the locations are segmented into 64 Kbyte blocks. Four 16-bit segment registers are used to define the segments. Accessing data residing in different segments requires the updating of the segment registers and this will hinder the operation of the device, as discussed in [74]. There are additional limitations of the chip that will degrade the performance of the multi-layer structure. The chip uses a multiplexed data/address bus structure, 16 lines are shared by both data and address busses. Such an arrangement demands extra hardware to latch the address generated by the chip when reading, or writing, data from, or to, the memory and it also reduces the efficiency of the system.

	Motorola MC68000	Intel 8086	Inmos T222
Address bus	23-bit	20-bit	16-bit
Memory Access (in bytes)	16M	1M	60 K(external)+ 4 K(on-chip)
Maximum Working Frequency	10MHz	8MHz	20MHz
Interface	Memory mapped	Memory mapped	4 serial links
Price (in pound)	7.3	14	29

Table 8.1 Parameters of the 16-bit microprocessors

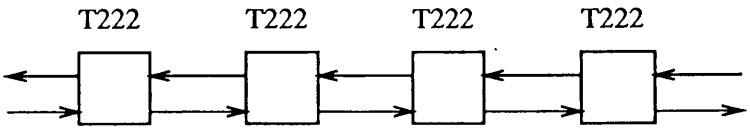
The transputer has two special features, namely the four serial links and fast on-chip memory. The T222 transputer is a 16-bit device which can operate at a maximum speed of 20MHz. The serial links enable a transputer to communicate with 4 other transputers in parallel at a maximum rate of 20 Mbit/s. The four serial links can be used for inter-element communication: two different configurations are possible, as

shown in Fig. 8.2. In the first configuration, each transputer communicates with an adjacent element via two links so that transmitting, or receiving, data from neighbours can be performed simultaneously. The second configuration enables each PE to collect information coming from its four nearest neighbours, with each link conveying data bi-directionally, and such an arrangement can improve operations involving data within a 5x5 window. However, data transfer via the serial links is restricted to a format requiring the sending of control bits to encode the data and this reduces the rate of data transfer to about 2 Mbytes/s. The T222 can address 64 Kbytes of memory partitioned into external and internal memory. There are 4 Kbytes of internal memory provided for high speed processing, whilst the external memory is used to store both data and programme codes.

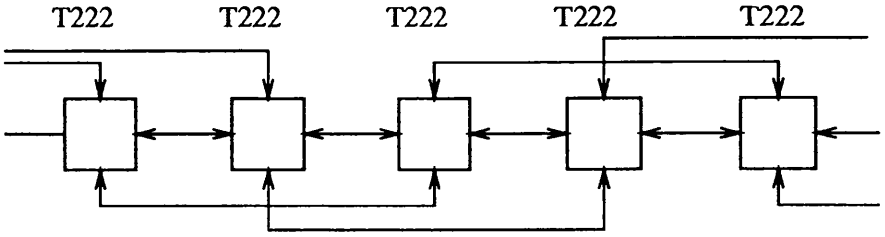
There are two disadvantages that render the T222 unsuitable for the proposed system. First, the limited address space: the T222 can only address 64 Kbytes of memory and this may not be sufficient for layer III's applications. Secondly, the transputer is relatively more expensive. The maximum functioning frequency of the T222 is 20MHz and in order to benefit from such a high operating frequency memory with short access time is required, this in turn will increase the cost of the system. As an example, 64 Kbytes of 45ns access time RAM costs about £80, whilst the same amount of memory with 100ns access time costs only £26.

The above discussion reveals that the Intel device is expensive and has other disadvantages including segmented memory space and multiplexed data/address lines. The Inmos transputer is expensive too and can only address 64 Kbytes of memory. The last device - the Motorola MC68000 - is cheaper than the other two devices studied. It can address up to 16 Mbytes of memory via 23 non-multiplexed address lines and includes 17 32-bit registers, which enhance its efficiency in performing 32-bit computations. The processor's assembly language [67] contains a set of 54 basic instruction types, including multiply and divide, which support five basic data types and 14 addressing modes. Because of these advantages, the MC68000 is chosen as the computing unit of the multi-layer system: the proposed architecture for the processing element in each layer is given in the following sections. First, however, some general features regarding the implementation of the MC68000 are discussed.

The microprocessor does not provide special I/O lines for communication with external devices. All external devices are mapped to the processor's address space. When the processor wants to read, or write, data from, or to, an external device, the address of that device will be generated. A simple decoding (or demultiplexing) circuit is connected to the address lines so that the proper control signals for the device



(a) Utilising two serial links for communicating with adjacent PEs



(b) Utilising the serial links for communication with four nearest neighbours

Figure 8.2 Different neighbourhood connectivities for a transputer array

can be generated from its corresponding address. The processor supports data in both byte and word formats so that data input/output devices can be either 8-bit or 16-bit wide.

8.2.3 The Layer I Processing Element

The function of layer I is to perform low-level operations on the input image. The input images have a resolution of 256x256x8-bits, which is equivalent to 64 Kbytes of information. The system is devised to operate on a continuous input of images, which implies that there is a substantial amount of data to be processed in each second. Besides the processing, the inputting and outputting of image data must also be effective otherwise the performance of the system will be degraded. An effective image I/O mechanism is one design requirement for this layer and, as concluded in Chapter 5 Section 5.4, an effective inter-layer element communication method and a unit that can perform functions similar to the binary gate are to be included.

There are six major components in each PE, namely the computing unit, local memory, two neighbour registers, an output register and the binary unit, as shown in Fig. 8.3. The computing unit is a MC68000 microprocessor with a 16-bit data bus and 23-bit address bus. Different components communicate via a 16-bit data bus and the 23-bit address bus is mainly used by the computing unit for addressing the local RAM. The unit will perform all the operations required by this layer. In the current design, the local memory consists of 64 Kbytes of RAM arranged in word format and 8 Kbytes of dual-ported memory which is provided as an image store. The dual-ported memory is accessed by the video I/O devices so that images can be directly loaded from, or read by, those devices. Such an arrangement will reduce the time required for image I/O and improve the efficiency of the system.

The RAM is used to store both data and programme codes and is loaded during system initialisation. The programme codes define the functions to be performed by the MC68000 processor. Communication between neighbouring elements is via the neighbourhood registers (RR and RL), which are 16-bit tri-state latches. The registers are connected together so that the output from one register is directly connected to the input of the register next in the chain, as shown in Fig. 8.4. Two 16-bit data channels are formed and data moves only in a single direction within each channel. The RL register is for passing data to a PE's left-hand side neighbour, whilst the RR register is for transferring data in the opposite direction. The outputs of the RR and RL registers are connected to neighbouring PEs' data bus and each PE can enable its neighbours' RR and RL registers for outputting their data. To transfer data from one PE to one of

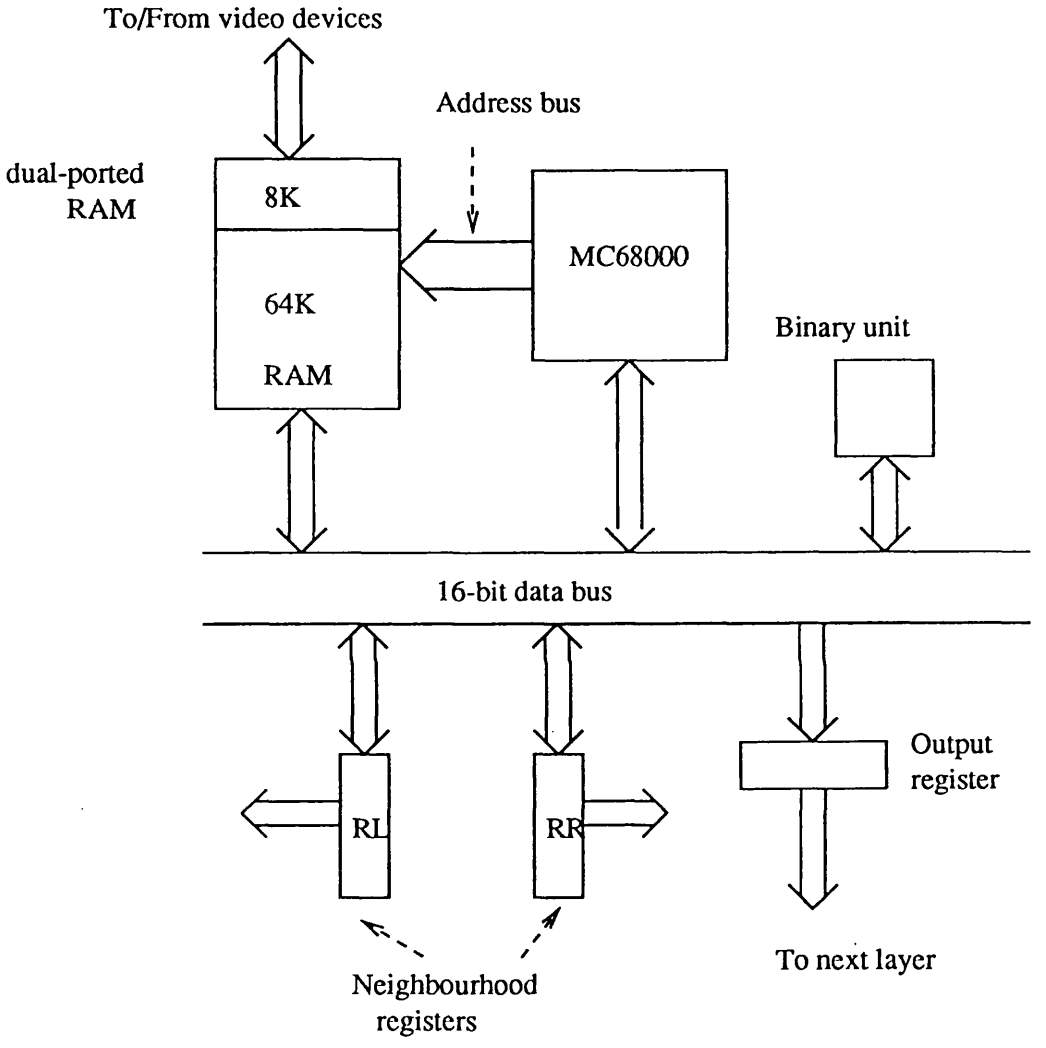


Figure 8.3 Block diagram of a layer I PE

its adjacent neighbours requires two operations. First, every PE loads its data into either the RL, or RR, register depending on the destination of the data. Second, each PE enables the corresponding adjacent PE's neighbourhood register to output the data, which is then read by the PE via its data bus. Two switches are provided at the ends of the channels and two connectivities, *wrap-around* and *reflect* (see Fig. 4.5), can be achieved by proper control of the switches. The movement of data within the channels can be controlled either by the layer I controller or the processing elements. When data from a PE is not transferred to its adjacent neighbours but to a PE farther away then the control of the data channel is taken by the layer controller. Besides communicating with elements in the layer, the channels are also used to transfer data between the layer controller and the array. The output register is used to communicate with elements in layer II, further description of which can be found in Section 8.3.

The binary unit can be regarded as a modified version of the binary gate (see Fig. 4.2) and is designed to generate an address according to the binary bit pattern of a pixel's eight nearest neighbours. As discussed in Chapter 5 Section 5.2.1, the address formed can be applied to different lookup table algorithms, the efficiency of which is dependent on the effectiveness of the mechanism to generate the address. The unit consists of three registers, as shown in Fig. 8.5. The registers North (N) and South (S) are 1-bit wide and have a function similar to the edge registers in the CLIP7A array, i.e. to store data coming from the North and South directions in a 3x3 window. Outputs of these registers are connected to register Addr, which also receives inputs from adjacent PEs' binary units and data busses. The address generated can be retrieved by reading the content of register Addr. The complete operation will take about 70 clock cycles, including loading data to the North/South registers and reading the address from the register Addr. Without the binary unit, performing the same function (i.e. generating an address from the nearest neighbours) will take about 300 clock cycles, including reading the data from a PE's neighbours via the data channels and performing the appropriate number of shift operations, as described in Chapter 5 Section 5.2.1.2. Because of its simplicity, the binary unit provides an efficient approach to generating an address from a PE's eight nearest neighbours.

8.2.4 The Layer II Processing Element

Layer II processing elements perform intermediate-level operations on data coming from layer I. The structure of a layer II processing element is depicted in Fig. 8.6. There are six major components in each element including the computing unit, memory unit, temporary storage, two neighbourhood registers and a communication

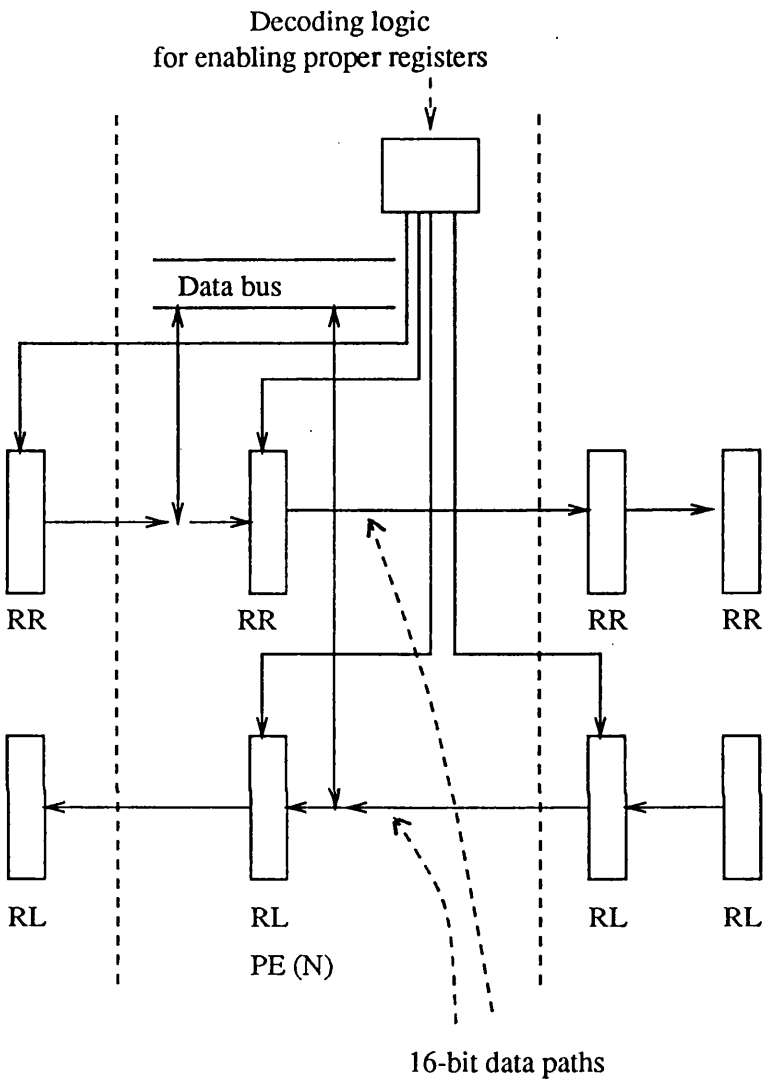


Figure 8.4 Intra-layer communication network

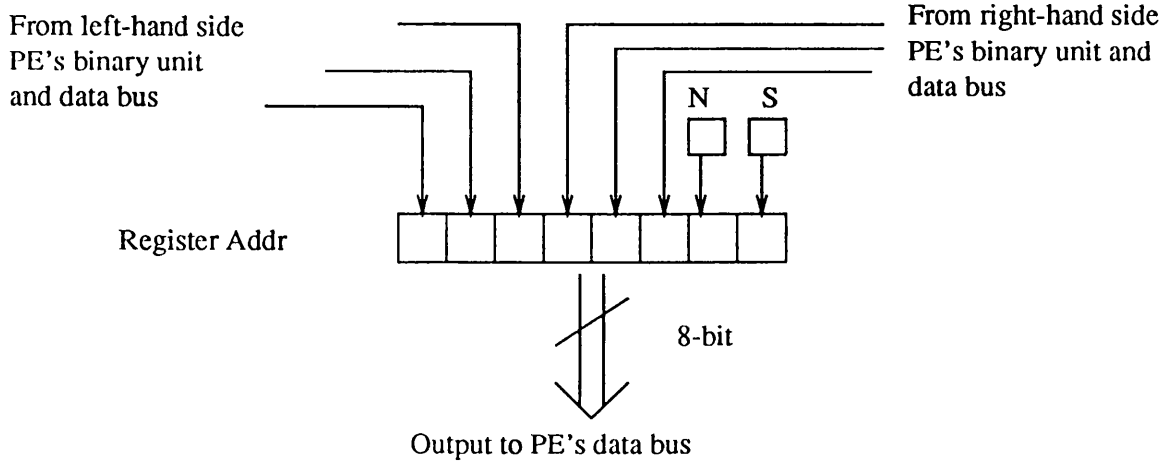


Figure 8.5 Block diagram of the binary unit

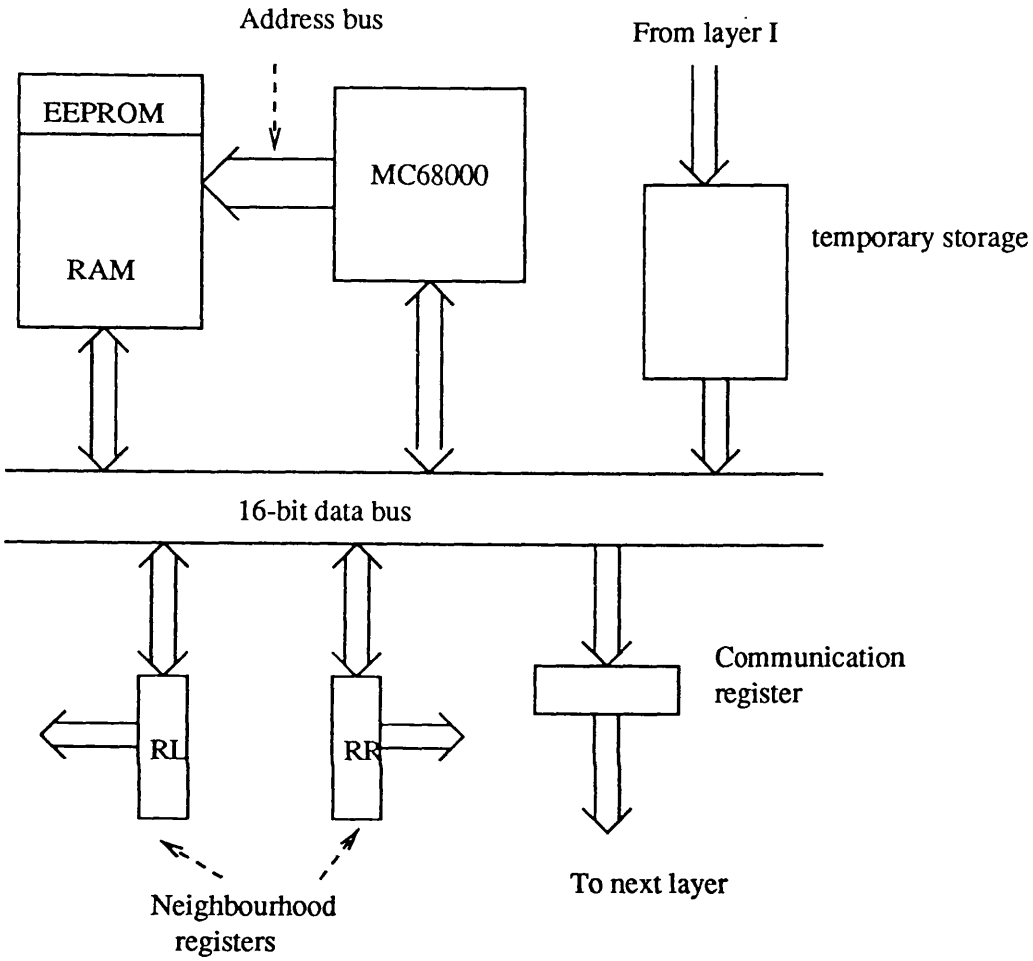


Figure 8.6 Block diagram of a layer II PE

register. All elements are 16-bit devices and they communicate through a 16-bit data bus.

The computing unit in each PE is a MC68000 microprocessor which will perform all the computation required in this layer. Operations of the computing unit are governed by programme codes stored in the local memory. There are 128 Kwords of local memory including both RAM and EEPROM. The function of the EEPROM is to accommodate special mathematical functions such as floating-point sub-routines, required by intermediate-level operations. In the present design, high-precision arithmetic is achieved by software, but if there is a demand for a faster solution then a hardware solution could be considered. A floating-point co-processor, e.g. the Motorola MC68881 [71], can be included in the processing element rather easily due to the architecture of the microprocessor. This is another advantage of using off-the-shelf components.

The two neighbourhood registers (tri-state latches) are used to communicate with elements within the layer. Two 16-bit data channels are formed by connecting the registers in chains, similar to those in layer I. Data only travels in a single direction in each channel and the connections between the channels are controlled by switches at both ends of the channels. Again *wrap-around* and *reflect* connectivity can be achieved. The data channels are also utilised for communication with the controller.

The communication register and temporary storage are devised for inter-layer communication. The communication register is utilised as an output buffer for data that is ready to be transmitted to layer III. The temporary storage is a First-In-First-Out (FIFO) register which holds data coming from layer I temporarily. A detailed description of the operations of the unit will be included in Section 8.3.1.

8.2.5 The Layer III Processing Element

The function of the layer III processing elements is to perform high-level processing on the data coming from layer II. The structure of layer III elements is very similar to those in layer II (see Fig. 8.7) except that there is no communication register in layer III because it is the last layer. The major components include a computing unit, a memory unit, a temporary storage and two neighbourhood registers. These communicate via a 16-bit data bus. The functions of the computing unit, memory unit, neighbourhood registers and temporary storage are identical to those in layer II.

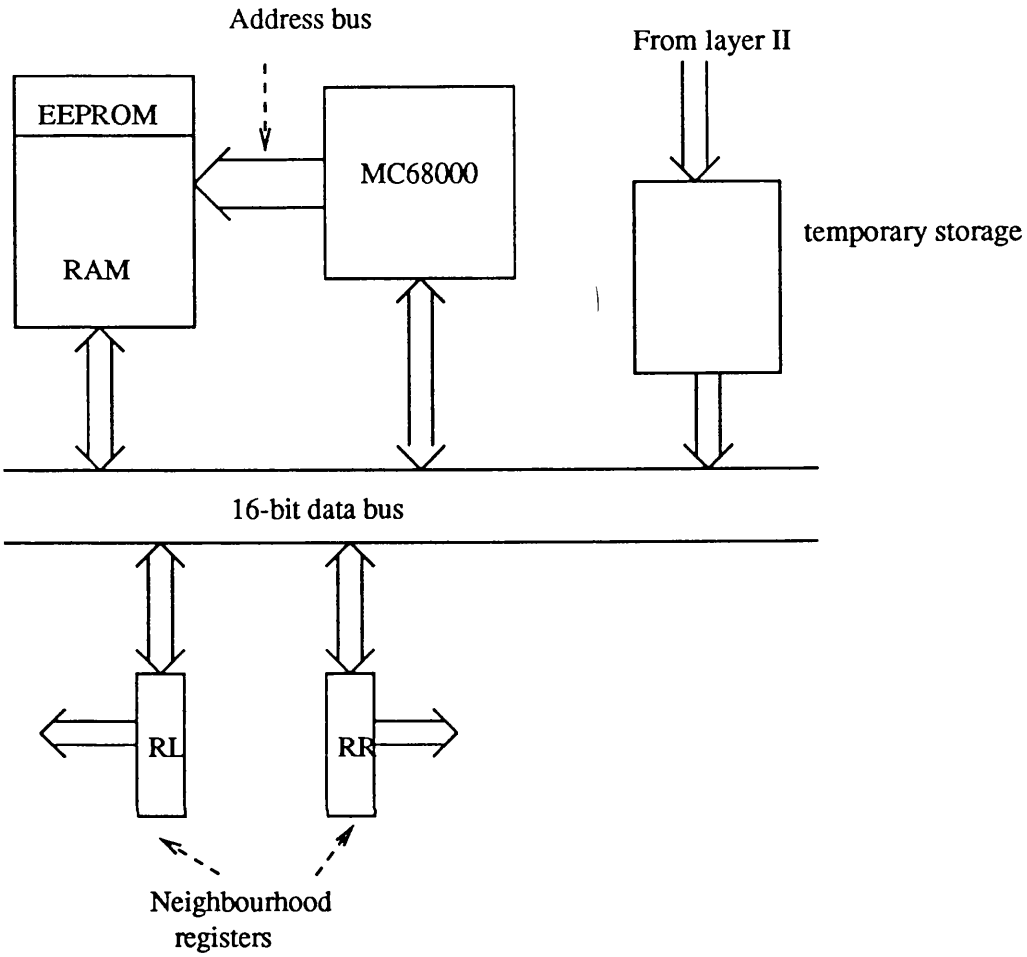


Figure 8.7 Block diagram of a layer III PE

8.3 The Inter-layer Communication Networks

In Chapter 7 Section 7.3, the structures of the inter-layer communication networks were described. The implementation of the communication networks is described in the following sections.

8.3.1 The Layer I to Layer II Communication Network

The communication network between layer I and layer II is the many to one to many mapping, shown in Fig. 7.3(d). The most important feature of the network is its grouping and distributing effect. All the data coming from layer I are first collected by a single processing unit called the dealer, which then distributes the data to layer II elements. The implementation of the network is shown in Fig. 8.8. Instead of using an extra device as the dealer, its operations are performed by the layer I controller.

When low-level operations complete, data ready to be transmitted from layer I to layer II are loaded to the communication register, see Fig. 8.3. The outputs of the communication registers are connected to a 16-bit communication bus, as shown in Fig. 8.8. As described in Chapter 7 Section 7.3.2.1, only the non-empty communication registers will be accessed, so a decoding logic is required to locate the non-empty registers. Instead of locating all non-empty buffers at the same instant, the devised logic (see Fig. 8.9) will only locate the first non-empty buffer, starting from the left-most element in the array (i.e. PE(0)). This is because only one buffer can be read at each instant. After the data from the first non-empty buffer is transferred to layer II, the buffer will be cleared so that the next non-empty buffer can be selected.

The important feature of the decoding logic is to enable only the first non-empty buffer and to disable the others. Once a buffer is enabled, its data will be loaded to the inter-layer communication bus. This is achieved by the generation of a propagation signal which indicates if the first non-empty buffer has been located. The propagation signal travels from left to right and is conveyed through the signal lines NL (Neighbour Left) and NR (Neighbour Right) in the decoding logic (see Fig. 8.9). The signal line NR is connected to the NL input of the logic next in the sequence, as shown in Fig. 8.9, so that the signal can be propagated along the row. Other signal lines included in the logic are the EF and EN. The signal EF (Empty Flag) represents the status of an output buffer, when an output buffer is empty then its EF signal is '0'. The line EN stands for ENable: when EN is high, data in the output buffer is loaded to the communication bus. Relationships between the input and output signals are presented in the form of a truth table as depicted in Table 8.2. The functions of the logic can be explained according to the status of the propagation signal, which is either in a state of

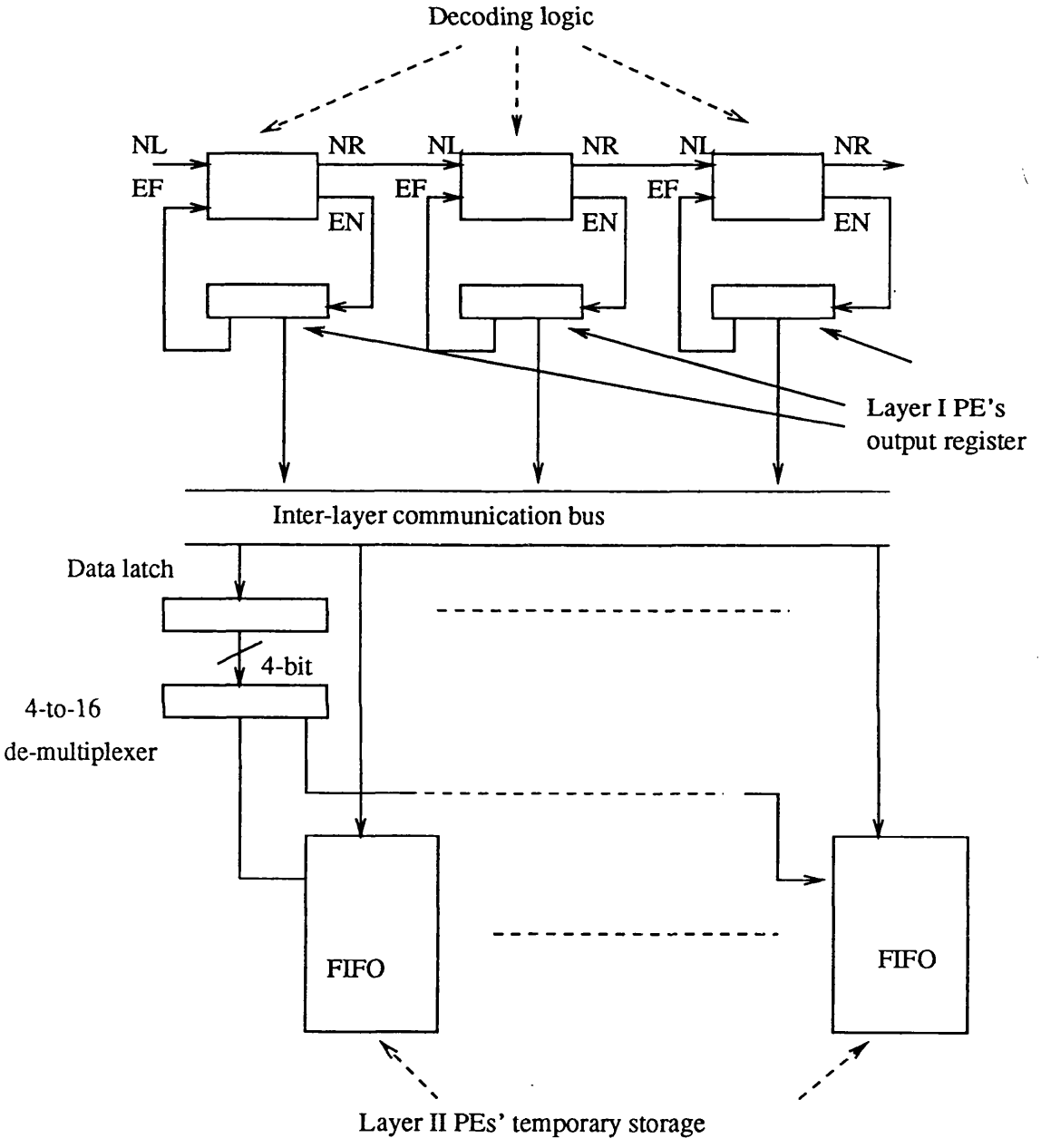


Figure 8.8 Block diagram of the layer I to layer II communication network

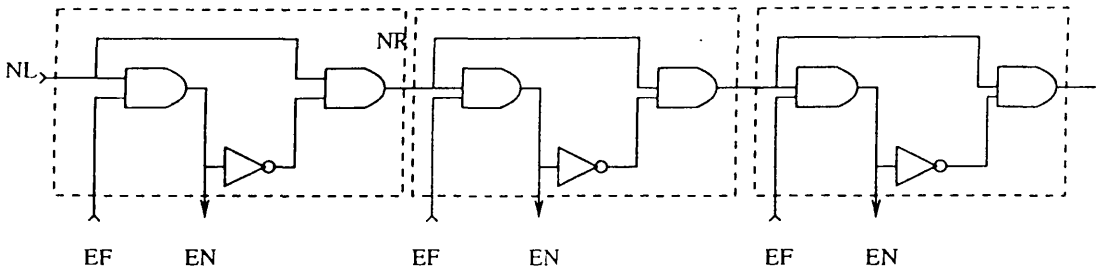


Figure 8.9 Structure of the decoding logic

disable or enable. When the signal is in a state of enable, i.e. NL is '1', then the first non-empty buffer (having EF '1') will be enabled and this in turn will alter the status of the propagation signal to disable by setting its NR output to '0'. The next logic in the sequence will then have a '0' in its NL input, which will disable the buffer regardless of the status of its EF input and output a '0' in its NR signal. Since the propagation signal is now in the state of disable, no other buffer can be enabled.

Input		Output	
EF	NL	EN	NR
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Table 8.2 Truth table for the decoding logic

When a register is enabled, its data is output to the communication bus, which is also connected to the temporary storages (FIFOs) of the layer II elements. Only one layer II element is enabled: exactly which one is enabled is determined by the value of the data being sent. As discussed in Chapter 7 Section 7.2.3.1, the data transferred from layer I to layer II are classified as symbols, each of which is encoded with a label, as shown in Fig. 7.5. The label value will identify symbols originating from the same object and is used to select a layer II element so that data belonging to the same object will be processed by a single layer II element. Since there are only 16 PEs in layer II, only the lower 4-bits of a label's value is needed to distinguish the layer II PEs. The least significant 4 bits of the communication bus are connected to a latch, the output from which drives a demultiplexer, as shown in Fig. 8.8. The demultiplexer will generate the signal to activate one of the 16 FIFOs so that data in the communication bus is loaded into the proper FIFO.

The FIFOs, as stated in Section 8.2.4, are used as temporary stores because they have two advantages:

1. Writing to, and reading from, a FIFO is internally sequential so that no address information is required. This simplifies the architecture of the bus controller (or

the dealer).

2. The FIFOs suggested can store 512 bytes of data so that data coming from layer I can be stored continuously and layer II elements can operate on the data previously received at the same instant. This will improve the performance of the system by enabling data transfer and processing to be carried out simultaneously.

8.3.2 The Layer II to Layer III Communication Network

Communication between layer II and layer III elements is through a 16-bit communication bus, as shown in Fig. 8.10. As discussed in Chapter 7 Section 7.3.3, each layer III element will contain a unique data file, which can either be a database or a set of models. Data derived from layer II is broadcast to all the elements in layer III so that model matching or database searching can be carried out in all layer III elements in parallel. The most important component of the network is a bus arbitration unit, which will administer the layer II elements so that only one of them can occupy the communication bus. The arbitration unit is included in the layer II controller, and besides ensuring only one layer II element is broadcasting data at each instant, the unit also generates signals to control the temporary stores (FIFOs) in layer III.

Each layer II element will generate a signal when it is ready to send its data. If there is only one element ready to send and the bus is not occupied then the arbitration unit will enable that element to take over the communication bus. If the communication bus is already occupied then the controller will signal the element to wait. When more than one element is ready to send at the same instant, the controller will only enable one element to send, whilst it signals other elements to wait. In such a case, the element nearest to the left has a higher priority. Once an element is enabled, it can write to the communication bus until all its data are transmitted.

Once data is available in the communication bus, the controller will issue a signal to enable the FIFOs in elements of layer III to be written. Using FIFO as a temporary storage can enable a layer II element to carry out high-level processing while inter-layer communication is in progress. This in turn will improve the overall performance of the system.

8.4 The Controlling Mechanisms and The Controllers

In the proposed three layer system the controlling mechanisms and the structures of the controller vary in each layer. The reason that different controlling mechanisms are used is due to the different computing requirements demanded at each layer. The

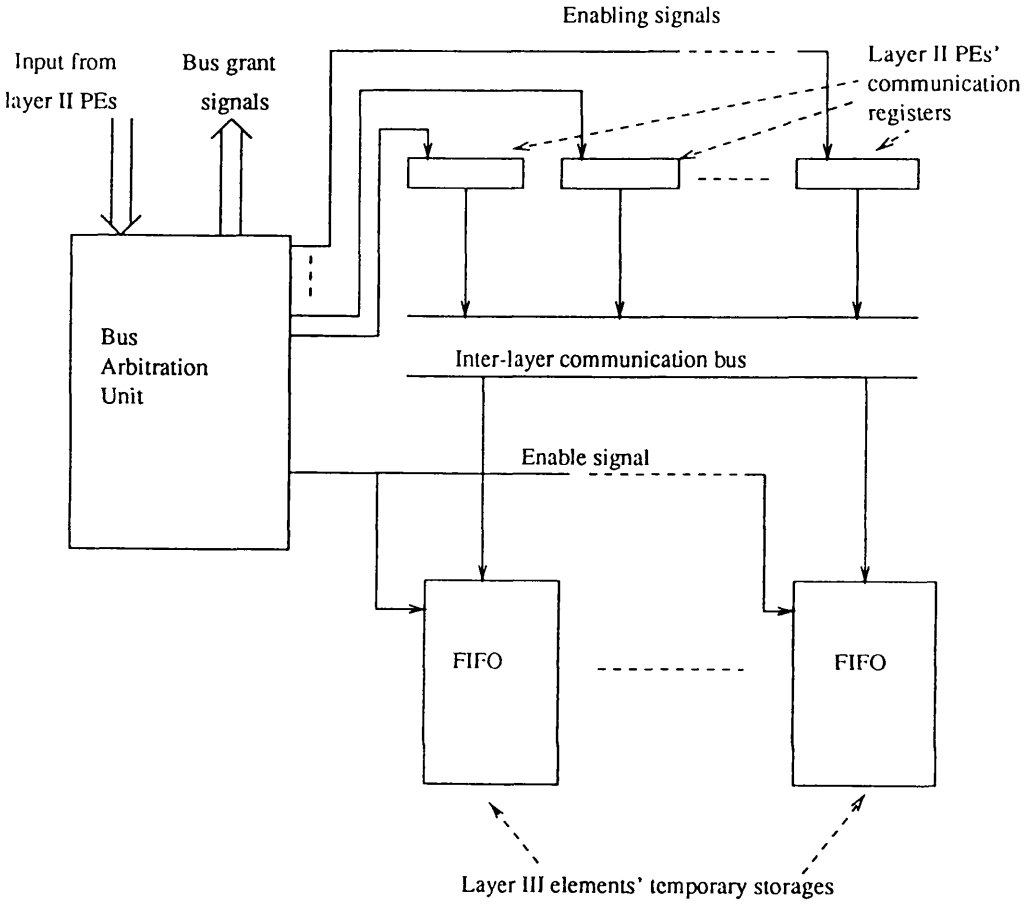


Figure 8.10 An overview of the layer II to layer III communication network

controlling mechanism employed by each layer is classified according to Flynn's taxonomy (see Chapter 1 Section 1.1), which uses the instruction stream and the data stream to categorise a system. Since each processing element employed in each layer has its own instruction store, the term single instruction stream can be used to describe such a linear array only if two conditions are fulfilled. First, each element must store the same instruction codes; secondly, the instructions have to be executed in the same sequence. If these conditions are not satisfied then the term multi-instruction will be applied.

Layer I performs low-level operations. In many low-level operations the identical function has to be applied to every pixel in an image and the SIMD controlling mechanism is appropriate for those operations. However, the efficiency of many operations can be improved by allowing conditional functions to be performed. Such a requirement can be expressed as the IF...THEN...ELSE construct. The thresholding operation, for example, can be implemented rather easily in such a construct:

IF (pixel value > thresholding level)
THEN (new value = 255)
ELSE (new value = 0)

In order to achieve this, each PE will receive the same programme codes and the sequencing of the codes in each element will basically be the same except during the execution of the conditional function. The controlling mechanism in this layer is, therefore, classified as SIMD with local function control (see Chapter 3 Section 3.2).

In layer II, each element will process set(s) of data belonging to one object (or several objects). Since each set of data represents a different entity (or object), the operation to be performed by each element may be governed by the data received. In this layer, each element still has the same instruction codes but their sequence of execution is determined by the input data. In such a situation each element will be executing a different section of codes and operating on different data, therefore the controlling mechanism of the layer is classified as MIMD. In layer III, each element processes data coming from two sources: internal and external. The internal data are stored in the local memory and represent the information of a database file. Because each PE will embody a different section of a database file, data coming from the internal data stream are different in each PE. The external data come from layer II. Every element receives the same set of data from the external source. Since there are two data streams and different data are provided internally, the layer is regarded as having multi-data streams. Since the sequencing of the programme codes will depend on

relationships between the external data and the internal data, this means that layer III is also categorised as a MIMD system.

There is a separate controller in each layer. The functions of a controller include acting as an interface between the host and a linear array, and to synchronise the operations of elements within an array when necessary. Additionally, controllers for layers I and II also administer the inter-layer communication networks. In order to achieve those requirements, it is proposed that the basic components of a controller should include a MC68000 microprocessor, local memory and two interface units, as shown in Fig. 8.11, all the components communicating via a 16-bit data bus.

The operations to be performed by the MC68000 are stored in its local memory, which consists of both RAM and ROM. The ROM will store the operations which should be executed during system initialisation. The RAM will be loaded with codes governing the operations of the controller in a particular application. The codes are loaded from the host during system initialisation.

Each controller has two interface units devised for communication between the host, and the linear array. The interface units for communicating with the host have the same structure, as shown in Fig. 8.12(a). The unit basically consists of two data stacks employed for temporarily storing data coming from, or leaving for, the host. The stacks are constructed from FIFOs, which are 16-bit wide.

The structures of the interface units for communicating with the arrays are different. This is because, except for layer III, extra circuitry is required for the control of the inter-layer communication network, as shown in Fig. 8.12(b). Basically, there are two data buffers for latching data coming from, or going to, the linear array. The buffers are connected to the communication channels of an array, so that data can be transferred to, or from, the array through the channels. In the layer I controller, the control logic will embody the demultiplexer, which generates signals to control the FIFOs in layer II's PEs (see Section 8.3.1). In layer II, it will include the bus arbitration unit as described in Section 8.3.2.

The flow of control in the system is bottom-up started from layer I. After the initialisation of the system, layer I will begin its operations. Layer II elements are triggered when data is available from layer I; layer III elements are initialised in a similar manner. The flow of operations is shown in Fig. 8.13. The host will be running a monitor programme in order to extract information returning from the system.

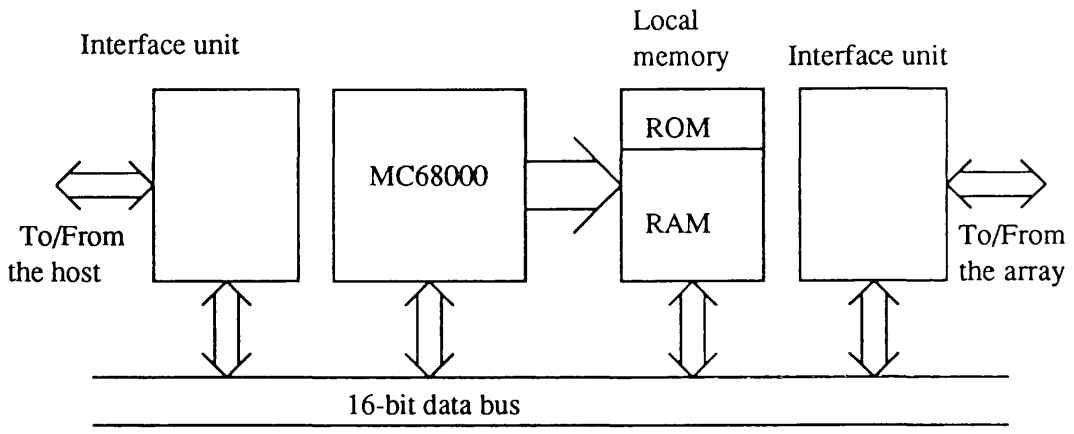
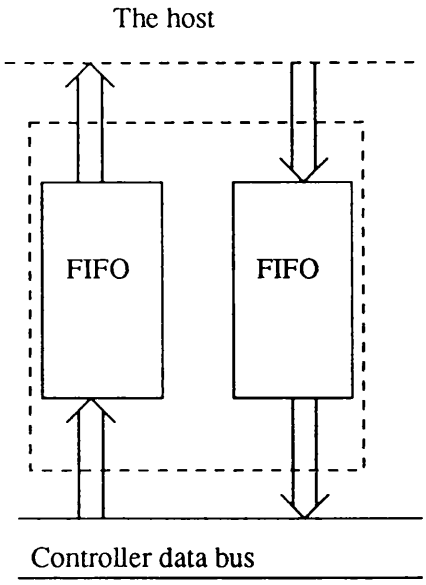
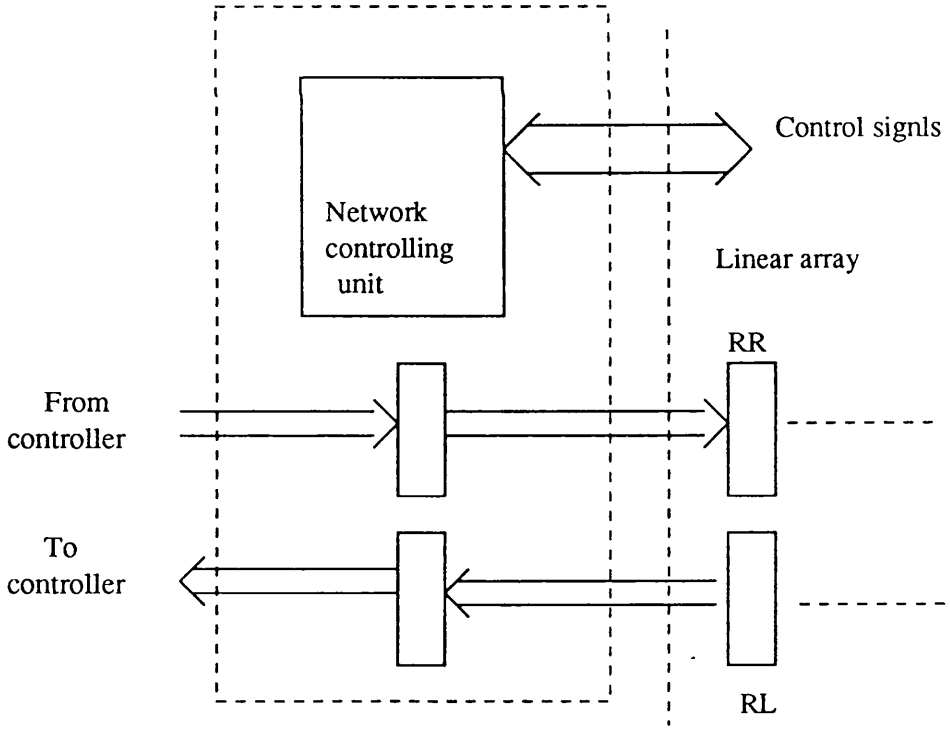


Figure 8.11 Block diagram of the array controller



(a) Block diagram of the host interface unit



(b) Block diagram of the array interface unit

Figure 8.12 Block diagrams of the host and array interface units

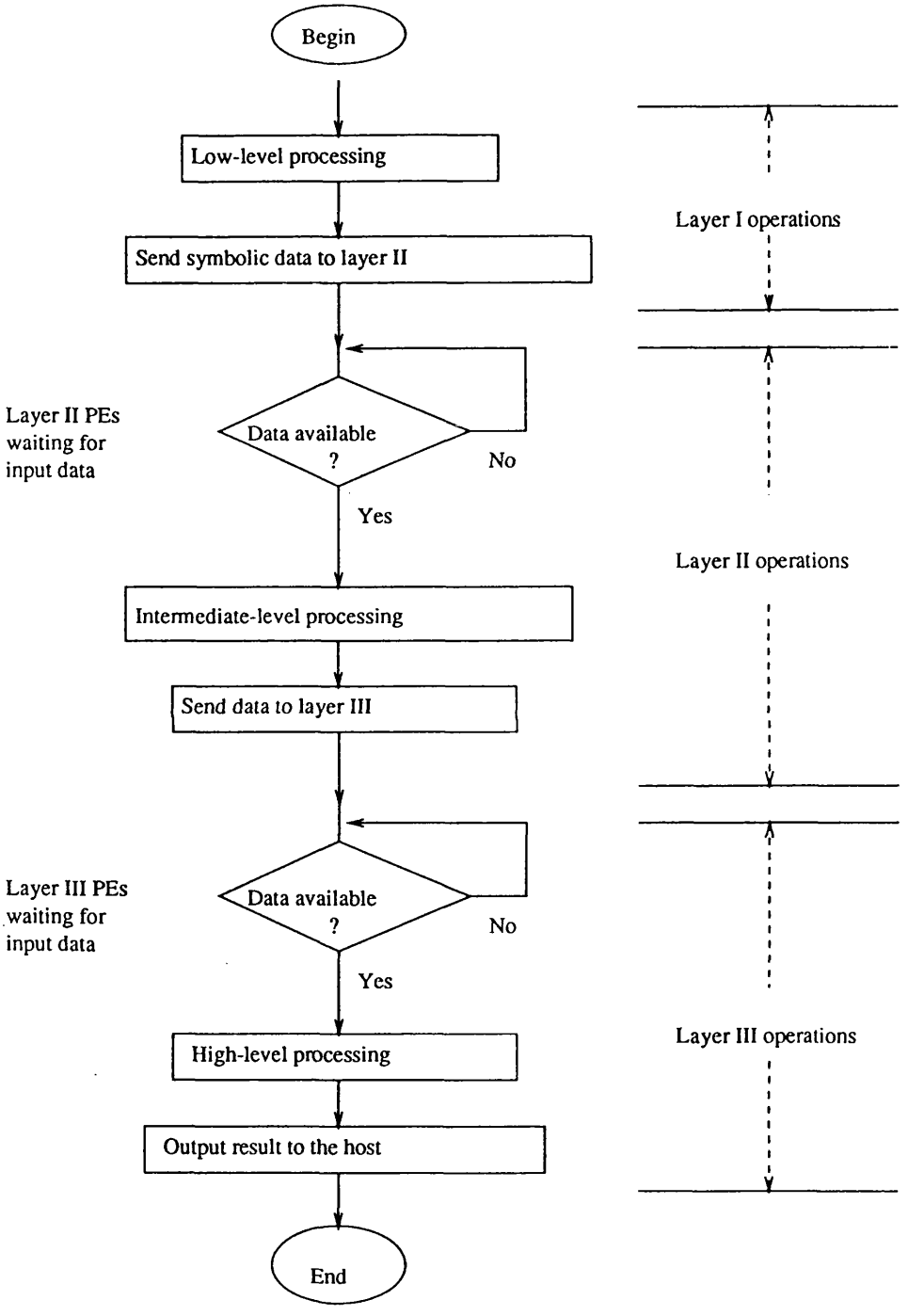


Figure 8.13 Flow of operations in the multi-layer system

8.5 System Software

In addition to the hardware components, the system software is also a very important issue in the system design. Development of the system software for the multi-layer system is a difficult task for two reasons:

1. The operations to be performed in each layer are different; and
2. Communications between layers must be programmed separately.

Since both the controlling units and the processing elements of the proposed system use the Motorola MC68000 microprocessor, its assembly language will be utilised as the lowest level programming language. At the most primitive stage, individual layers and controllers will be programmed separately.

From a user's point of view such a programming model may be very inconvenient because different programmes have to be written. Moreover, assembly language programming requires a very good understanding of the properties of the processor. A higher level or more user-friendly programming methodology should then be developed. In order to achieved this, a hierarchy of software is conceived with pseudo-codes for each level presented in Fig. 8.14.

The lowest level of the hierarchy will be the MC68000 assembly language. Operations to be performed by each layer, and by each controller, are written in separate programmes. The second level will enable a user to programme the system by some high-level language, such as the C language. In this level, there are three sub-level programming languages, each of which is devised for programming a linear array and its controller. Each sub-level language will have a similar construction. Operations performed during inter-layer communication will be implemented as subroutines and embodied in a layer's programming language. The second level programming language for the layer I array will be similar to the C7VM language. Special data types will be available in this level, for example the data type Vector could be included.

In level three, operations to be performed by each layer will be programmed in a single language, as shown in Fig. 8.14. The language will embody all the features provided in level two. Operation performed by a particular layer is identified with a suitable prefix.

At the moment only three programming levels are suggested but additional levels are possible. Level four, for example, could provide some pre-defined functions which may involve operations at more than one layer. A more user friendly programming method, like the YAM [50], could be developed upon level four.

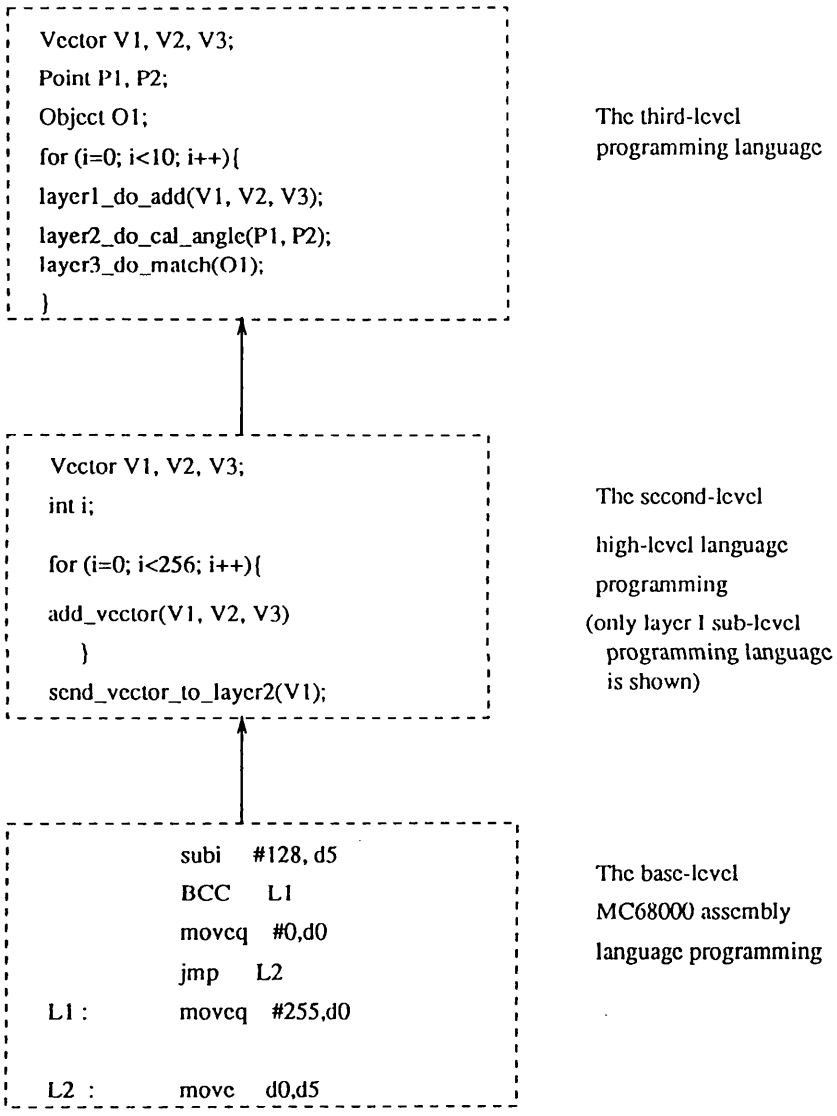


Figure 8.14 Pseudo-codes for the envisaged system software

8.6 Estimated Performance of Some Operations

In the above sections, both hardware and software aspects of the proposed multi-layer system have been introduced. Before concluding this chapter, the processing time required by the system to carry out some operations at all levels are listed in Table 8.3. The results are compared with those of the CLIP7A system. Execution time is estimated in terms of the number of clock cycles that are required to carry out the operation - assuming that the multi-layer system operates at 10MHz and the CLIP7A system at 5MHz. Operations performed in layers II and III are more problem dependent and therefore three commonly applied integer computations are examined. The low-level operations refer to the processing of an image with 256x256 8-bit pixels.

The performance of a system in carrying out different operations is determined by the computing power of its processing elements. The computing unit (the MC68000) employed by the proposed system is not a particularly fast device, therefore, the performance of the system for some operations is not superior than the CLIP7A system. However, the multi-layer system will be 10 times faster than the CLIP7A system in some operations for the following reasons:

1. The addressing mechanism of the CLIP7A system is less effective as only a 8-bit address bus is provided for the PEs.
2. The instruction set of the CLIP7A chip is very limited, making the programming of the system for complex operations difficult and inefficient.
3. The CLIP7A chip is a special purpose processor and inefficient when performing multiply, divide and signed integer arithmetic. It takes 273 clock cycles to multiply two 8-bit integers, for example, whilst the MC68000 can complete a 16x16-bit multiplication in 70 clock cycles. This degrades the system's performance in intermediate and high level operations.

As the system is designed to tackle vision problems involving different processing levels by providing a structure which maps well to the problem, it is more important to consider the improvement achievable by such a structure and this is discussed in the next chapter.

8.7 Summary

In this chapter, four features of the proposed system have been introduced, including the structure of the processing elements, the inter-layer communication networks, the controlling requirements and the system software. The objective of this

Type	Operation	Multi-layer	CLIP7A
Low-level (Image operation)	Add	0.5ms	1.7ms
	Logical OR	0.5ms	1.7ms
	Thresholding	1.2ms	4.4ms
	Sobel operators	8.1ms	72ms
Intermediate or high level (for a single PE) (see explanation below)	Gradient	20μs	174μs
	$(Distance)^2$	20μs	212μs
	Compare	3μs	60μs

Gradient - to compute the gradient of a straight line formed by two points
$(Distance)^2$ - to compute the square of the distance between two points
Compare - comparing two values including an error factor

Table 8.3 Estimated computation time for operations in different levels

Of course the individual instruction times do not tell the complete story. The way in which the times quoted in Table 8.3 affect the overall system performance was considered in the following chapter.

chapter has been to provide an overall description of the structures of various components embodied in the multi-layer unit. The design of the system is based on the results derived from chapters 5 and 6.

The system will be constructed from off-the-shelf components in order to save both development time and cost. The processing elements will employ the Motorola MC68000 microprocessor as the computing unit because of its low cost. Since the identical computing device is used in different layers, this simplifies both the controlling and programming requirements.

The layer I array is under SIMD control with functional autonomy. In order to enable high speed image I/O, each PE embodies 8 Kbytes of dual-ported memory, which can be accessed directly by the image I/O devices, as an image store. A binary unit has been devised to generate an address according to a pixel's 8 nearest neighbours' bit pattern. Elements in the layer communicate via two 16-bit channels and this also applies to elements in layers II and III.

Both layer II and III are MIMD arrays and their PEs have a similar structure. EEPROM is provided to accommodate special mathematical subroutines, which are required by intermediate and high level operations. The PEs employ FIFO as a temporary storage of data coming from other layers. The FIFOs are controlled by the inter-layer communication devices and this allows a PE to continue its operations, while inter-layer communication is in progress.

Two inter-layer communication networks have been described and special hardware devices are required to realise the networks. To enable efficient data transfer from layer I to II, a decoding logic which locates the first non-empty data buffer in the layer I array is designed. Because a different data passing mechanism is employed by the network between layer II and III, this necessitates a bus arbitration unit. The unit will generate proper signals to control both layer II and III elements during inter-layer communication.

There is a separate control unit for each layer. Its functions include acting as an interface between the host and a layer, and to synchronise the operations of elements within an array when necessary. Additionally, controllers for layers I and II also administer the inter-layer communication networks. In order to achieve the control requirements, each controlling unit also embodies a MC68000 microprocessor.

The basic programming language for the system will be the assembly language of the MC68000, which is employed by the arrays and their controllers. In order to provide a more efficient, or user-friendly, programming methodology a hierarchy of system software having three levels is envisioned. The estimated performance of some operations is presented.

Chapter 9
Summary and Discussion

9.1 Summary

The objective of this work has been to develop a parallel computing system that will deal with problems in image processing which include operations at low, intermediate, and high level. The project advanced in three phases, namely, background studies, investigation of design criteria and system design. From the background studies, it was concluded that a multi-layer architecture providing both spatial and functional parallelism would satisfy the processing requirements demanded by the problems. The multi-layer architectures studied in Chapter 2 are classified as pyramid systems and the 2-D array is utilised as the basic structure. Pyramid systems are further divided into homogeneous and heterogeneous. The heterogeneous approach could be adopted for the envisioned system but it was found to be costly and complex to build. Because the linear array can be applied to both image based and non-image based processing, it was proposed as the basic architectural unit. Additionally, linear arrays have greater flexibility and are easier to construct.

The design of a multi-linear-array system began by studying different linear arrays, followed by investigations in two areas :

1. Design criteria for linear arrays.
2. Matching the architecture of the system to the structure of the problem.

The investigations came to concentrate on four related issues:

1. The effectiveness of local autonomy.
2. The structural properties of linear arrays.
3. Processing requirements for the image processing problems.
4. The relationships between the structure of the problem and the architecture of the system.

The CLIP7A system was utilised as a representative linear array and different algorithms, designed so as to examine the relevant topics, were implemented on CLIP7A. The results obtained led to a conceptualised design of the multi-linear-array system.

The envisioned system consists of three layers performing, independently, low-level, intermediate-level and high-level processing, respectively. The number of elements provided in each layer is proportional to the amount of data anticipated: there are 256 elements in layer I, 16 in layer II and 4 in layer III. Elements in layer I operate

on the input image. Symbols, representing different entities appearing in the image, are extracted and processed by layer II elements. Ideally, each layer II element operates on a single entity (an object or a pattern) but if more than 16 entities appear in an image then elements operate on several of them. If there is a single entity in the input image then elements can operate on different entities coming from different frames (assuming that there is a continuous input of images). Elements in layer III collect all the available information from layer II and carry out appropriate operations in order to derive an answer to the problem.

As different data types are processed at each layer, the design of the inter-layer communication networks was affected. The networks were determined on the basis of three factors, namely efficiency, expandability and the ease with which fault tolerance could be achieved. A many-to-one-to-many communication network was provided between layers I and II, with the network enabling elements in layer II to collect symbolic data belonging to the same entity. Between layers II and III a communication bus is provided, with a suitable controlling mechanism for the bus envisaged so as to enable layer II elements to broadcast data to elements in layer III.

All processing elements in the proposed system employ the same 16-bit microprocessor as their computing unit, namely the Motorola MC68000. Such a design simplifies both controlling mechanism and programming requirements. All three arrays have dual inter-element communication channels. Each layer has its own controller - also based upon a MC68000 microprocessor. Functions of the controllers include synchronising the operations of the array, exchanging data between the host and the array, and, if necessary, controlling the inter-layer communication network. The different computing requirements mean that the controlling mechanism in each layer is different: layer I under SIMD control, with local function control, and layers II and III under MIMD control.

The last component discussed was system software. The basic programming language is the MC68000 assembly language, though a more user-friendly programming language is envisaged.

9.2 Discussion

The design of the multi-layer system was based on the problem structure depicted in Fig. 6.1. Based on the data types evolved at each level of processing, the communication requirements between elements in different layers are determined. In Chapter 7, estimation of the inter-layer communication time was presented. Using the benchmark image as an example, the inter-layer communication time was found to be

about 0.63 ms, which approximates to 2% of the processing time (40 ms) for a frame of an image in a video-rate application. Inter-layer communication time is dependent on the quantity of data to be transmitted from one layer to another and will clearly vary with different applications. Since the estimated time is low compared with the video frame time, it is concluded that the proposed networks provides an efficient mechanism for transferring data between layers, even when the quantity of data to be transmitted is higher than anticipated. The effectiveness of the networks is achieved by matching their structures to those of the problems and reducing the rate of intra-layer communication. The latter requires the support of special hardware - the decoding logic (see Chapter 8 Section 8.3.1). Due to its simplicity it can be implemented quite easily without imposing a penalty on the cost of the system. Fault tolerance and expansion can also be easily achieved in the proposed networks, thereby improving the flexibility of the system.

The major difference, macroscopically speaking, between the proposed system and other linear image processors is the extra processing layers provided. Each layer performs its operations upon the input data independently. If there is a continuous input of images then a pipelining effect, with each layer operating simultaneously, is achieved. It is, therefore, important to examine how the performance of the system is enhanced by the extra processing layers, comparing with a single layer system. The base processing layer of the proposed system is used as an example single layer system. This is because many single layer systems are devised for low-level processing, which is the major function of the base layer. Moreover, the layer embodies many special features, including an image store, addressing autonomy and dual-communication channels, which make it a flexible device.

Before determining the performance of the proposed system, the combined cost of the middle and top layers is estimated. Let us suppose that the base layer costs C_1 pounds and the combined cost of the extra layers is C_2 pounds. Because of the similarity between elements in layers II and III, it is assumed that processing elements in those layer have the same cost. The PEs in both layers II and III embody extra memory (both ROM and EEPROM) and the FIFO. Due to the costs of these components, it is assumed that a PE in those layers is about three times as expensive as a PE in layer I. Since there are a total of 20 PEs in layers II and III, C_2 should be about 25% of C_1 (because there are 256 PEs in layer I). This implies that the proposed system is 25% more expensive than the example linear system.

The processing time required to derive an answer from the input image can be represented as 3T seconds. If processing operations performed by both systems at

each level take the same period of time (T) to complete then with a continuous input of images, the multi-layer system can output an answer every T seconds. It will take 3T seconds for the single layer array and in this case, the multi-layer system will be three times faster than a single layer system for 25% extra cost. Obviously, there are many other situations which, relating to the variation in processing time between levels, should be considered.

First, assume that the physical configuration of the system is fixed, i.e. the multi-layer system will cost 25% more than a single layer system. Equation (9.1) represents the total processing time required to solve a problem by the single layer array.

$$T=T_1+T_2+T_3 \tag{9.1}$$

where T is the total processing time; and T_1, T_2, and T_3 are the respective processing times of the low, intermediate and high level layers. If the corresponding layer in the multi-layer system also takes the same processing time (T_1, T_2, T_3) to complete its operations then the multi-layer system can produce an answer to the problem every T' seconds (assuming that there is a continuous input of images). Because the inter-layer communication time is relatively short, it can be neglected. T' is the maximum value among T_1, T_2 and T_3 (because of the pipelining effect of the multi-layer system). In an ideal situation, as discussed above, $T_1 = T_2 = T_3$. The ratio $\frac{T}{T'}$ represents the order of improvement achievable by the multi-layer system as opposed to a single layer system. If $\frac{T}{T'} > 1.25$ then the multi-layer system is effective (because it is 25% more costly). For problems having a high $\frac{T}{T'}$ ratio, the proposed system provides a cost-effective solution. In the original benchmark problem [75], the computing time required for both low and high level processing is in the same order of magnitude, and is greater than that of intermediate-level processing. This implies the $\frac{T}{T'}$ ratio in such a situation is about 2, i.e. the multi-layer system will be 2 times faster than a single layer system.

In the above discussion, it was assumed that computing time required in each level of processing is the same for both systems. This assumption may not be appropriate to intermediate and high level operations because the spatial parallelism (i.e. the number of PEs) provided by the single layer array is higher than the multi-layer system (256 PEs are available in the single layer array comparing with 16 and 4 provided by the multi-layer system). A more detailed study between the two systems is necessary.

The cost of the single linear array is considered first. In order to carry out intermediate and high level processing, extra hardware is required (for example extra memory is required to store both data files and programme codes) and this will increase the cost of the system. If only extra memory is provided then a 25% increment is assumed and the two systems have the same cost. Now, the processing time for intermediate-level operations is examined. The single layer system has 256 PEs, comparing with 16 in the multi-layer system, but it is unlikely that the system will be 16 times faster because of the nature of intermediate-level processing. Intermediate-level processing involves the grouping of symbolic data, originated from a unique object, residing in PEs along the array. Parallelism is achieved by assigning one PE to operate on data coming from the same object. If there are less than 16 objects present in an input image then the extra PEs provided by the single layer array will not participate in any processing. Moreover, in order to collect information from different PEs, the amount of inter-element communication is large (as demonstrated by an example in Chapter 7) and this will generate a substantial overhead to the processing time. Because of the above, the assumption that the processing time required for intermediate-level operations is the same for both systems stands.

High-level operations involve the gathering of information from PEs representing different objects and the production of an answer to the problem by referring to some data files. Again, the gathering of information from different PEs requires a large amount of inter-element communication, this in turn imposes an overhead to the processing time. However, the parallelism achieved by allowing each PE to carry a unique data file and performing high-level processing independently will be substantial. This implies that the single layer array will be faster than the multi-layer system in performing high-level operations. The level of improvement depends on the problem and it is assumed that an order of 10 is possible. Based on the above analysis, the processing time required by the single layer array (T_S) and the multi-layer system (T_M) for solving a vision problem is represented by equation (9.2) and (9.3).

$$T_S = T_1 + T_2 + T_3 \tag{9.2}$$

$$T_M = T_1 \text{ or } T_2 \text{ or } 10T_3 \tag{9.3}$$

where T_1, T_2 and T_3 are the respective processing times of low, intermediate and high level operations. Because both systems have the same cost, the multi-layer system is more effective when the ratio $\frac{T_S}{T_M} > 1$. This is true when either T_1 or T_2 is greater than $10T_3$. If the processing time T_1, T_2 and $10T_3$ are in the same order of magnitude then $\frac{T_S}{T_M}$ is about 2, implying that the multi-layer system is almost 2 times faster than a

single layer system. On the other hand, if T_1, T_2 and T_3 are equal (or in the same order of magnitude) then the multi-layer system is less effective (the $\frac{T_S}{T_M}$ ratio is about 0.3). In such a case, expanding the top layer should reduce the processing time ($10T_3$) but, this in turn will increase the cost factor. In this example, if extra elements are added to layer III (can be easily achieved in the proposed structure) so that the original processing time ($10T_3$) is reduced to T_3 then the $\frac{T_S}{T_M}$ becomes 3. If layer III is expanded by a factor 10 then the multi-layer system will be 40% more expensive. However, comparing with the improvement in the processing speed, such an expansion is still cost-effective. Because the configuration of the system can be modified (by expanding, or reducing, the middle or top layer), this enables the effectiveness of the multi-layer system to be retained in different situations. Based on the above analysis, it is concluded that the multi-layer system will be a effective device for solving vision problems which comprise processing at each level. A more accurate analysis of the situation would require further consideration of specific examples.

Simulation of the system, generated by software or hardware, will be useful in collecting information in respect of the system's performance in various situations. This will assist the designer to determine under which situations extra elements should be provided without affecting the system's cost-effectiveness. Additionally, simulation can be utilised to investigate other aspects of the system, such as the inclusion of other levels of local autonomy and a top-down controlling mechanism. The proposed layer I design embodies local function control and the performance of the system may be improved with other levels of autonomy (see Chapter 3 Section 3.2). Because each PE in the layer includes its own programme store, other levels of autonomy (e.g. local algorithm control) can be implemented easily. In the present design only bottom-up control is envisaged but other multi-layer systems employ top-down control, and the original benchmark problem clearly implied the need for it. One way to implement top-down control is via the layer controllers: by providing communication links between the controllers a request for operations can be sent from the controller in the upper layer to the target layer's controller, which will then issue control signals to trigger elements within that layer. Similarly, data can also be transferred from the top layers to the base layer via the communication links between the controllers.

In the next stage of the development procedure, simulation of the proposed system should be designed and an investigation of the above topics then carried out.

References

[1] R.C. Gonzalez and P. Wintz, Digital Image Processing (1st ed), Addison-Wesley (Reading, Mass.) 1977.

[2] F.H. Cheng, W.H. Hsu, and M.Y. Chen, Recognition of Handwritten Chinese Characters by Modified Hough Transform Techniques, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 11, No. 4, April 1989, pp. 429-439.

[3] C.L. Wan, K.W. Dickinson, A. Rourke, M.G. Bell, Z. Xu, and N. Hoose, Low-cost Image Analysis for Transport Applications, Proc. Image Analysis for Transport Applications, IEE Digest No. 1990/035, February 1990, pp. 1/1-1/18.

[4] M.J. Flynn, Very High Speed Computer Systems, Proc. of IEEE, Vol. 54, No. 12, December 1966, pp. 1901-1909.

[5] H.J. Siegel, L.J. Siegel, F.C. Kemmerer, P.T. Mueller, Jr., H.E. Smalley, Jr., S.D. Smith, PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition, IEEE Trans. on Computers Vol. C-30, No. 12, December 1981, pp. 934-947.

[6] A.P. Reeves, Survey: Parallel Computer Architectures for Image Processing, Computer Vision, Graphics, and Image Processing 25, 1984, pp. 68-88.

[7] R. Duncan, A Survey of Parallel Computer Architectures, IEEE Computers Vol. 23, No. 2, February, 1990, pp. 5-16.

[8] M.J.B. Duff, CLIP4: A Large Scale Integrated Circuit Array Parallel Processor, Proc. 3rd Int. Joint Conf. on Pattern Recognition, 1976, pp. 728-733.

[9] D.J. Hunt, The ICL DAP and Its Application to Image Processing, in Languages and Architectures for Image Processing, edited by M.J.B. Duff and S. Levialdi, Academic Press (London) 1981, pp. 275-282.

[10] K.E. Batcher, Design of a massively parallel processor, IEEE Trans. on Computers, Vol. C-29, No. 9, September 1980, pp. 836-840.

[11] A.P. Reeves and R. Rindfuss, The BASE 8 Binary Array Processor, IEEE Proc. PRIP 79, August 6-8, 1979, Chicago, Illinois, pp. 250-255.

[12] P. Gemmar, H. Ischen and K. Luetjen, FLIP: A Multiprocessor System for Image Processing, in Languages and Architectures for Image Processing, edited by M.J.B. Duff and S. Levialdi, Academic Press (London) 1981, pp. 245-256.

[13] S.L. Tanimoto, Architectural Issues for Intermediate-level Vision, in Intermediate-level Image Processing, edited by M.J.B. Duff, Academic Press (London) 1986, pp. 3-17.

[14] M.J.B. Duff, Some Considerations on the Limitations of Image Processing Computer Architectures, in Proc. IAPR Workshop on Computer Vision - Special Hardware

and Industrial Applications, October 12-14, 1988, Tokyo, Japan.
[15] J.L. Basille, P. Dalle and S. Castan, Iconic and Symbolic Use of a Line Processor, in Intermediate-level Image Processing, edited by M.J.B. Duff, Academic Press (London) 1986, pp. 231-241.
[16] D.H. Schaefer, P. Ho, J. Boyd, C. Vallejos, The GAM Pyramid, in Parallel Computer Vision, edited by L. Uhr, Academic Press (Boston) 1987, pp. 15-42.
[17] S.L. Tanimoto, T.J. Ligocki, R. Ling, A Prototype Pyramid Machine for Hierarchical Cellular Logic, in Parallel Computer Vision, edited by L. Uhr, Academic Press (Boston) 1987, pp. 43-83.
[18] V. Cantoni, S. Levialdi, PAPIA: A Case History, in Parallel Computer Vision, edited by L. Uhr, Academic Press (Boston) 1987, pp. 3-13.
[19] A. Merigot, P. Clermont, J. Mehat, F. Devos and B. Zavidovique, A Pyramidal System for Image Processing, in Pyramidal Systems for Computer Vision, edited by V. Cantoni and S. Levialdi, Springer-Verlag (Berlin) 1986, pp. 109-124.
[20] P. Clermont, A. Merigot, Real Time Synchronisation in a Multi-SIMD Massively Parallel Machine, Proc. Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, Oct. 5-7, 1987, Seattle, Washington, pp. 131-136.
[21] S.P. Levitan, C.C. Weems, A.R. Hanson, E.M. Riseman, The UMass Image Understanding Architecture, in Parallel Computer Vision, edited by L. Uhr, Academic Press (Boston) 1987, pp. 215-248.
[22] C.C. Weems, A.R. Hanson and M. Riseman, The Architectural Requirements of Image Understanding with Respect to Parallel Processing, to be appeared in IEEE Proc. 1991.
[23] G.R. Nudd, R.M. Howarth, T.J. Atherton, N.D. Francis, G.J. Vaudin and D.W. Walton, A Heterogeneous Architecture for Parallel Image Processing, Proc. of UK Information Technology 88 Conference, Swansea, 4-7 July 1988, pp. 495-499.
[24] G.J. Vaudin, G.R. Nudd, T.J. Atherton, S.C. Clippingdale, N.D. Francis, R.M. Howarth, D.J. Kerbyson, R.A. Packwood, D. Walton, A Generalised Parallel Architecture for Image Based Algorithms, Eurographics'89, Hamburg, September 1989.
[25] Connection Machine Model CM-2 Technical Summary, Thinking Machine Technical Report HA87-4, April 1987.
[26] W.D. Hillis, The Connection Machine, MIT Press (Cambridge, Mass.) 1985.
[27] E.W. Kent, M.O. Shneier and R. Lumia, PIPE (Pipelined Image-Processing Engine), Journal of Parallel and Distributed Computing 2, (1985), pp. 50-78.
[28] F.A. Briggs, K. Hwang, K.S. Fu and B.W. Wah, PUMPS Architecture for Pattern Analysis and Image Database Management, IEEE Computer Society Workshop on

Computer Architecture for Pattern Analysis and Image Database Management, Hot Springs, November 11-13 1981, pp. 178-187.

[29] S.Y. Kung, VLSI Array Processors, Prentice-Hall (Englewood Cliffs., N.J.) 1988.

[30] M.J.B. Duff, Pyramids - Expected Performance, in Pyramidal Systems for Computer Vision, edited by V. Cantoni and S. Levialdi, Springer-Verlag (Berlin) 1986, pp. 59-73.

[31] T.J. Fountain, The Use of Linear Arrays for Image Processing, Proc. Int. Con. on Systolic Arrays, San Diego, California, 1988, pp.183-192.

[32] R. Negrini and R. Stefanelli, Fault-tolerance Techniques in Arrays for Image Processing, in Pyramidal Systems for Computer Vision, edited by V. Cantoni and S. Levialdi, Springer-Verlag (Berlin) 1986, pp. 373-392.

[33] T.J. Fountain, Introducing Local Autonomy to Processor Arrays, in Machine Vision: Algorithms, Architectures, and Systems, ed. H. Freeman, Academic Press (Boston) 1988, pp. 31-56.

[34] L.A. Schmitt and S.S. Wilson, The AIS-5000 Parallel Processor, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 10, No. 3, May 1988, pp. 320-330.

[35] R.M. Lea, ASP: A Cost-effective Parallel Microcomputer, IEEE Micro, October 1988, pp. 10-29.

[36] T.J. Fountain, K.N. Matthews and M.J.B. Duff, The CLIP7A Image Processor, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 10 No. 3 March 1988, pp. 310-319.

[37] A.S.K. Lau, The C7VM System Programming Guide, Image Processing Group Internal Report No. 88/6, University College, London, 1988.

[38] B. Stroustrup, The C++ Programming Language, Addison-Wesley (Reading, Mass.) 1986.

[39] The NPL Linear Array Processor: Technical Details, National Physical Laboratory, 1982.

[40] B. Lindskog, PICAP3: An SIMD Architecture for Multi-dimensional Signal Processing, Ph.D Thesis, Linköping University, Sweden (1988).

[41] K.N. Ngan, A.A. Kassim and H.S. Singh, Parallel Image-Processing System Based on the TMS32010 Digital Signal Processor, IEE Proc. Vol. 134, Pt. E, No. 2, March 1987, pp. 119-124.

[42] A.L. Fisher, P.T. Highnam, T.E. Rockoff, Architecture of a VLSI SIMD Processing Element, IEEE Int. Con. on Computer Design: VLSI in Computers and Processors, Oct. 5-8 1987.

[43] J.L. Basille, S. Castan and J.Y. Latil, Système Multiprocesseur Adapté au

Traitement d'Images, in Languages and Architectures for Image Processing, edited by M.J.B. Duff and S. Levialdi, Academic Press (London) 1981, pp. 205-213.

[44] M. Annaratone, E. Arnould, T. Gross, H.T. Kung, M. Lam, O. Menzilcioglu and J.A. Webb, The Warp Computer: Architecture, Implementation, and Performance, IEEE Trans. on Computers, Vol. C-36, No. 12, December 1987, pp. 1523-1538.

[45] C. Rieger, ZMOB: Doing It in Parallel, IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Hot Springs, November 11-13 1981, pp. 119-124.

[46] K.N. Matthews, The CLIP7 Image Analyser - A Multi-Bit Processor Array, Ph.D Thesis, University College, London, 1986.

[47] K.N. Matthews, An Introduction to the Mik Microcode Assembler, Image Processing Group Internal Report No. 86/2, University College, London, 1986.

[48] K. Beurle, The Vcode Manual, Image Processing Group Internal Report No. 88/4 E&OE, University College, London, 1989.

[49] S.F. Javid and D.G. Greenwood, The DIMPL Library, Image Processing Group Internal Report No. 89/3, University College, London, 1989.

[50] S.F. Javid and D.G. Greenwood, YAM User Manual, Image Processing Group Internal Report, University College, London, 1990.

[51] S.F. Javid, 3-D Image Processing by Voxel, Ph.D Thesis, University College, London (1991).

[52] C.C. Foster and T. Iberall, Computer Architecture (3rd ed.) Van Nostrand Reinhold (New York) 1985.

[53] C. Arcelli, L. Cordella, S. Levialdi, Parallel Thinning of Binary Pictures, Electronics Letter, Vol. 11, No. 7, April 1975, pp. 148-149.

[54] Y-S Chen and W-H Hsu, A Modified Fast Parallel Algorithm for Thinning Digital Patterns, Pattern Recognition Letters 7 February 1988, pp. 99-106.

[55] G.P. Otto, Propagation in Cellular Arrays, in Cellular Logic Image Processing, edited by M.J.B. Duff and T.J. Fountain, Academic Press (London) 1986, pp. 41-68.

[56] K.A. Clarke, Reconstruction of Nuclear Medicine Images on the CLIP4 Computer, Ph.D. Thesis, University College, London, 1984.

[57] C. Kornfeld, The Image Prism: A Device for Rotating and Mirroring Bitmap Images, IEEE Computer Graphics and Applications, Vol. 7, No. 5, May 1987, pp. 21-30.

[58] D.J. Potter, Analysis of Images Containing Blob-like Structures Using An Array Processor, Ph.D. Thesis, University College, London, 1984.

[59] C.J. Wu and J.S. Huang, Human Face Profile Recognition by Computer, Pattern

Recognition, Vol. 23, No. 3/4, 1990, pp. 255-259.
[60] L.D. Harmon, M.K. Khan, R. Lasch and P.F. Ramig, Machine Identification of Human Faces, Pattern Recognition Vol. 13, No. 2, 1981, pp. 97-110.
[61] G. Nagy, Chinese Character Recognition: A Twenty-Five-Year Retrospective, Proc. 9th Int. Conf. on Pattern Recognition, Nov. 14-17 1988, Rome, Italy, pp. 163-167.
[62] C.H. Leung, Y.S. Cheung and Y.L. Wong, A Knowledge-Based Stroke Matching Method for Chinese Character Recognition, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-17, No. 6, November/December 1987, pp. 993-1003.
[63] C. Weems, E. Riseman, A. Hanson and A. Rosenfeld, IU Parallel Processing Benchmark, Proc. Conf. Computer Vision and Pattern Recognition, Michigan, June 5-9 1988, pp. 673-688.
[64] A. Rosenfeld and A.C. Kak, Digital Picture Processing (2nd ed.), Vol. 2, Academic Press (New York) 1982.
[65] H.J. Siegel, Interconnection Networks for Large-Scale Parallel Processing, Lexington Books (Lexington Mass.) 1985.
[66] The TTL Data Book, Vol. 1, Texas Instruments 1983.
[67] M68000 8-/16-/32-Bit Microprocessors User's Manual, 6th ed., Prentice Hall (Englewood Cliffs., N.J.) 1989.
[68] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985, The Institute of Electrical and Electronics Engineers, 1985.
[69] D.A. Fraser, R. Gregory, B. Hawken, B. Holdsworth, G. Jones and M. Ryal, Introduction to Microcomputer Engineering, Ellis Horwood (Chiester) 1985.
[70] The Transputer Databook, 1st ed., INMOS, Bath Press (Bath) 1988.
[71] M68000 Family Reference, Motorola, 1988.
[72] D.A. Nicole, Reconfigurable Transputer Processor Architectures, in Multiprocessor Computer Architectures, edited by T.J. Fountain and M.J. Shute, North Holland (Amsterdam) 1990, pp. 227-244.
[73] A.J. Dent, A.R. Macdonald, C.F. Packer and C.F. Price, A Multi-Transputer Processor for High Level Image Processing Algorithms, Proc. Transputers for Image Processing Applications, IEE Digest No. 1989/22, February 1989.
[74] M68000 vs. iAPX86 Benchmark Performance, Motorola.
[75] C. Weems, E. Riseman, A. Hanson and A. Rosenfeld, The DARPA Image Understanding Benchmark for Parallel Computers, Journal of Parallel and Distributed Computing, Vol. 11, No. 1 January 1991, pp. 1-24.