THE DESIGN AND DEVELOPMENT OF HIGIPS FOR THE REAL-TIME HIGH-LEVEL IMAGE PROCESSING AND ANALYSIS

A Dissertation
Presented for the
Doctor of Engineering

Feng-Hui Yao
August 1992

Department of Electric, Electronic and
Computer Engineering, Faculty of Engineering
Kyushu Institute of Technology

To the People's Republic of China

ACKNOWLEDGMENTS

More than six years passed since I came to Kyushu Institute of Technolc Kitakyushu, Japan. My graduate school years have been enjoyable and hard. I would l to express my appreciation to all of those people who made it possible for me to fir this dissertation. In particular, I am very grateful for the unconditional support Professor K. Kato and Assistant Professor A. Tamaki, whose guidance and concern w essential to the successful completion of this work. I appreciate Associated Professoı Ishikawa, Professor S. Murakami, Professor H. Anzai, Professor K. Kurosu, Professoı Matsuoka, Professor M. Iwane, Professor E. Kawaguchi, and professor T. Yamashita their valuable and important discussions and comments. I am also thankful for the gr warm care from the clerk T. Uehara and other staffs in K.I.T., who provided a big c venience and a good environment for this project. Thanks are also due to all the mas course students in Kato laboratory for their help in my study.

I would also like to appreciate Kyushu Institute of Technology to enroll me and g me a chance to pursue the advanced study. Moreover, I would like to give my gratitude the Ministry of Education of Japanese government to award me a scholarship for whole graduate school. Additional thanks are given to my family for their encouraging to finish the graduate school studies.

ABSTRACT

Image processing has been widely used in medical engineering, remote sensing, television broadcast, industrial manufacturing. These applications demand not only static image processing but also dynamic image processing, understanding and analysis. To answer these demands, many high-speed high-performance computers and LSI chips specialized for image processing have been developed.

On the other hand, accompanying with advancement of the semi-conductor technology, the price of LSI chips is getting dramatic cheap. Furthermore, after breaking through the software crisis in 1960's, the programming environment has been being improved in the following sides. (1) Various of high-level languages have been developed; (2) Many syntax oriented editors easily to handle have been elaborated; (3) Programs can be reusable by making them into library routines; (4) Conversational debugging tools have been built on.

Under the condition of these abundant software and hardware resources, this dissertation analyzed the present situation of image processing and image processing systems. On the basis of this analysis, it suggested a new computer architecture for image processing and analysis from low-level to high-level. Basing on this suggestion, a prototype machine named as **HIGIPS** is designed and developed. Moreover, four kinds of low-level image processing such as the calculation of gray-level histogram, sharpening, smoothing, and edge detection have been performed. The usefulness of the architecture of **HIGIPS** is discussed.

Chapter 1 is a prologue of this dissertation. It gives the concept and the situation of parallel processing, the significance of this research, and the construction of this dissertation.

Chapter 2 classified image processing and image processing systems, and analyzed their situations. On the basis of this study, it suggested a new computer architecture for image processing and understanding from low-level to high-level.

Image processing is classified into (1) low-level processing, (2) intermediate-level processing, and (3) high-level processing.

Image processing systems are classified into (1) fully parallel processor, (2) local parallel processor, (3) pipeline processor, and (4) multiprocessor. The advantages and the disadvantages of these four kinds of systems are analyzed, especially for the disadvan-

tages of the pipeline processor and multiprocessor, a new computer architecture for image processing and analysis from low-level to high-level is conceived. This new computer architecture is the combination the pipeline architecture and multiprocessor architecture, i.e., the system in whole employs the pipeline architecture and each stage of this pipeline system employs the multiprocessor architecture. The machine with this new architecture is named as **HIGIPS**. **HIGIPS** is the abbreviation of High-speed General-purpose Image Processing System. It employs **N**×**M** processors, where **N** is the number of pipeline stages and **M** is the number of processors in a pipeline stage.

Chapter 3 related the design concept of the prototype **HIGIPS** in comparison with the traditional systems. The design of this prototype **HIGIPS** has the following features.

(1) Commercially obtainable general-purpose 16-bit microprocessor (V50) is selected as its **PE** (Processing Element);
(2) Combination of global memory and distributed local memory is used to improve the memory efficiency;
(3) Message exchanges among **PE** is performed via the global bus;
(4) Pipeline performance of the whole system is realized with the use of the specially designed dual port time-sharing memory module;
(5) Popularized general-purpose personal computer (NEC PC-9801) is adopted as the system controller;
(6) Parallel interface (GP-IB) is used as the interface between the system controller and **HIGIPS**.

Chapter 4 introduced the development of the prototype **HIGIPS**. The follow items are carefully considered during the development.

(1) Only off-the-shelf components are used. This shortened the developing time;
(2) By using a modular design concept, only two different boards, a CPU board — KIT-TA1 and a memory board — KIT-TA2, are implemented. This makes the construction of the whole system easy;
(3) Logic circuits are substituted by PAL devices. This decreased the area of the two different boards related in (2).

Chapter 5 related the image processing scheme on **HIGIPS** and its monitor facilities. The image processing scheme is as follows.

(1) To develop the processing program on system controller with C language;
(2) To segment the processing program into **N** sections, where **N** is the number of the pipeline stages;
(3) To download the executable file of each section to each stage of **HIGIPS**;
(4) To start the assigned processing according to the message from the system controller.

Monitor program of **HIGIPS** has the following facilities.

(1) Program down-load;
(2) Program execute;
(3) Program stop;
(4) Message exchange.

Chapter 6 showed the implementation of the **HIGIPS** monitor, and image processing experiment results on it. **HIGIPS** monitor is constructed by the following library functions.

(1) Memory write library function;
(2) Memory read library function;
(3) I/O write library function;
(4) I/O read library function;
(5) Program down-load library function;
(6) Program move library function;
(7) Program execute library function;
(8) Start-up test library function;
(9) Finish library function.

The corresponding commands to above library functions on system controller are prepared. These library functions and commands are developed with high-level programming language C. IPL (Initializing Program Loader) is developed with assemble language (55 machine instructions).

On this prototype **HIGIPS**, four kinds of low-level image processing such as the calculation of gray-level histogram, sharpening, smoothing, and edge detection are performed. Around **N**×**M** times speed-up is obtained. Therefore, the usefulness of the architecture of **HIGIPS** is confirmed.

Chapter 7 related the conclusions and listed the future works.

TABLE OF CONTENTS

LIST OF FIGURES

Figure *Page*

Figure *Page*

LIST OF TABLES

Table *Page*

LIST OF PHOTOS

Chapter 1

INTRODUCTION

As the use of computers affects increasingly the world economy in the broad fields, many problems to which people apply computers grow continually larger and more complex. Demands for faster and larger computer systems increase steadily. Fortunately, the technology base for the last twenty years has continued to improve at a steady rate—increasing in capacity and speed while decreasing in cost for performance. However, the demands outpace the technology. This raises the question, can we make a quantum leap in performance while the rate of technology improvements remains relatively constant?

Computer architects have followed two general approaches in response to this question. The first uses exotic technology in a fairly conventional serial computer architecture. This approach suffers from manufacturing and maintenance problems and high costs. The second approach exploits the parallelism inherent in many problems. The parallel approach seems to offer the best long-term strategy because, as the problem grows, more and more opportunities arise to exploit the parallelism inherent in the data itself.

Where can we find the inherent parallelism and how do we exploit it ? Most computer programs consist of a control sequence (the instructions) and a collection of data elements. Large programs have tens of thousands instructions operating on tens of thousands of or even millions of data elements. We can find opportunities for parallelism in both the control sequence and in the collection of data elements.

In the control sequence, we can identify threads of control that could operate independently on different processors. This approach, known as *control parallelism*, is used for programming most multiprocessor computers. The primary problems with this approach are the difficulty of identifying and synchronizing these independent threads of control.

Alternatively, we can take advantage of the large number of independent data elements by assigning one processor to each data element and performing all operations on the data in parallel. This approach, known as *data parallelism*, works best for large amounts of data. For many applications, it proves the most natural programming approach, leading to significant decrease in execution time as well as simplified programming.

Early examples of parallel processing machines are ICL's Distributed Array Processor (DAP) [37], Goodyear's Massively Parallel Processor (MPP) [8 ,9], and others. Each of

these machines has some elements of the desired architecture, but lacks others. For example, the MPP has 128×128 processing elements arranged in a two-dimensional grid, but interprocessor communication is supported only between neighboring processors.

This dissertation first analyzes the advantages and disadvantages of these conventional parallel processing machines. On the basis of this analysis, it then suggests a new architecture to perform the real-time high-level image processing, analysis and understanding. And third, it introduces the design, implementation and evaluation of a prototype machine named as HIGIPS (abbreviation of the High-speed General-purpose Image Processing System) which takes use of this new architecture.

A case study of the present image processing situation and an analysis of image processing machines is done in Chapter 2. Basing on this analysis, a new computer architecture for image processing is proposed. Chapter 3 discusses the design concepts with this new architecture. Chapter 4 explains the implementation of the prototype HIGIPS machine. Chapter 5 relates the programming environment of HIGIPS and the construction of the monitor program. Chapter 6 shows the evaluation of the prototype HIGIPS and the experiment results. Chapter 7 relates conclusions via this research and lists the future works.

Chapter 2

A CASE STUDY AND AN ANALYSIS OF TRADITIONAL IMAGE PROCESSING SYSTEMS

Nowadays, image processing has been widely applied in medical engineering, remote sensing, television broadcast, and industrial manufacture. These applications demand not only the static image processing but also the dynamic image processing, especially the real-time processing. To meet this demand, many architectures and parallel processing machines specialized on image processing have been proposed and realized [1, 4, 10, 16, 42, 43]. The following relates the taxonomy of image processing and image processing machines.

2.1 Classification of Image Processing

"Seeing" is governed by a complex mechanism having three primary components: environment (or matter) to be seen, radiant energy, and observer. Vision involves acquisition of radiant energy, formation and processing of imagery, and perception of the physical reality. Vision is investigated by three different schools of the scientific community. *Neurophysiologists* attempt to understand how sensor and neural mechanisms of biological systems function. *Perceptual Psychologists* try to understand the psychological issues governing the task of perception, and *Computer Vision Scientists* investigate the computational and algorithmic issues associated with image acquisition, processing, and understanding.

The mechanism governing light capture in biological and artificial systems is relatively simple and well understood. Development of eyeglasses, microscopes, cameras, and other sophisticated optical gadgetry are evidence of our thorough understanding of light capture and image formation. On the other hand, the problem of how to perceive objects in the three-dimensional (3-D) environment from the captured two-dimensional (2-D) images is not well understood in both.

A hierarchy of computational tasks utilized in a typical computer vision is presented in Fig.2.1 [96]. If a processing is based on arrays of numerical data that are in registration with or correspond directly to image data, then it is low-level. A processing based on symbolic descriptions of extracted image events, or view-specific symbolic instantiations of stored models and knowledge are intermediate level. A processing that is view- or scene-independent is high-level [100].

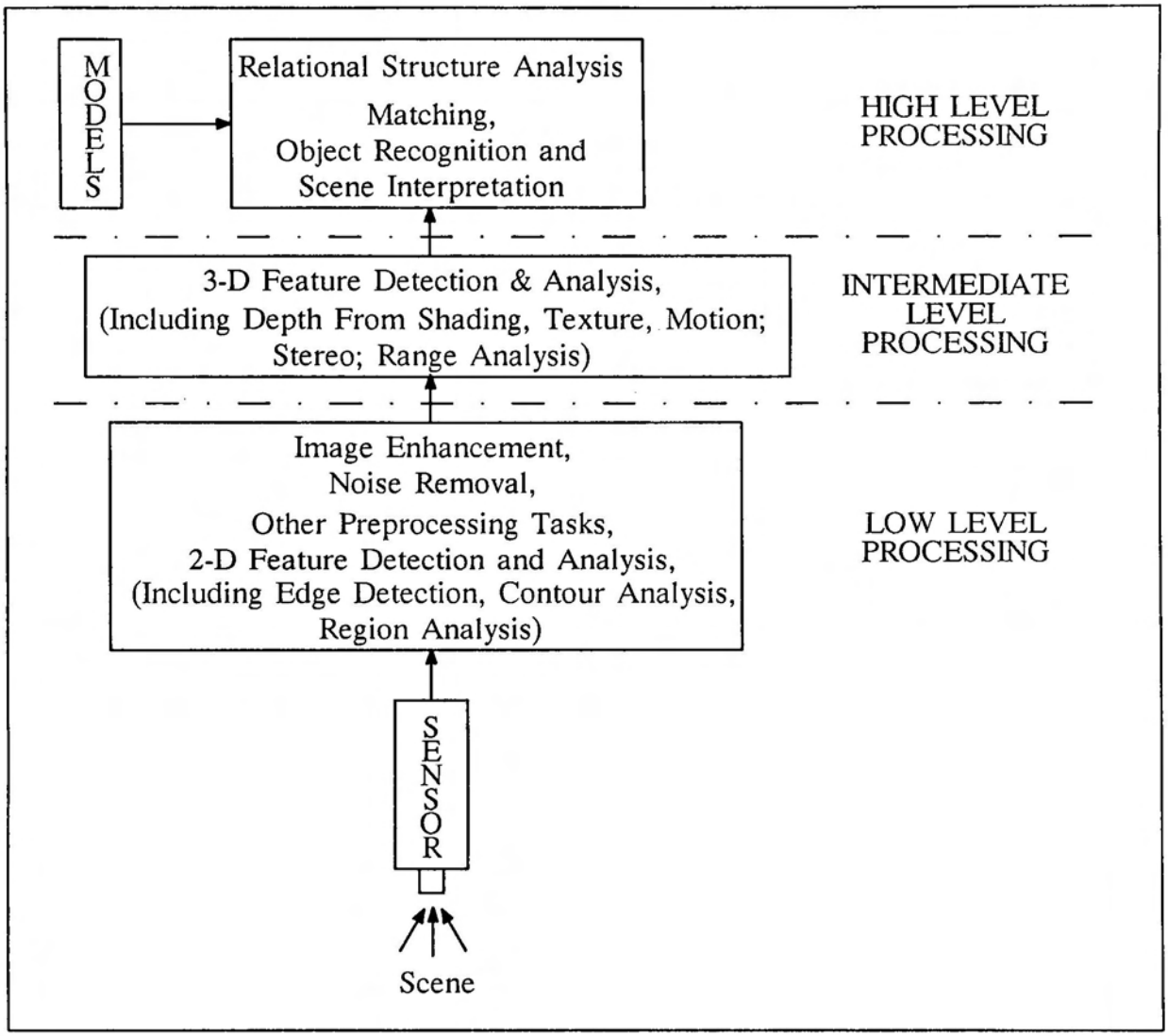

Fig.2.1. Hierarchy of computational tasks involved in machine vision.

Low-level processing deals with image enhancement, noise removal, or similar image processing functions. It also involves extraction and analysis of either edge-based or region-based features. Edge-based techniques exploit the properties of dissimilarities between pixels whereas region-based approaches attempt to capture similarities or homogeneity in region properties. This processing results in partitioning of the input image into its meaningful parts. These parts are known as blobs, and they need to correspond closely to various physical objects or their components appearing in the scene. These blobs are analyzed to derive useful features from spectral and spatial domains. Some of the commonly used features include gray level (multispectral/color), texture, size and shape measures.

Intermediate level processing mainly deals with the extraction and analysis of 3-D features from imagery. In some applications, such as high altitude remote sensing or document analysis, 3-D considerations are not required.

High-level processing involves final scene interpretation carried out by object recognition result procedures that analyze the results of intermediate level processing, and can be analyzed in a model-based paradigm. In this level, image features are back-projected to determine what object surfaces may have given rise to them. This approach proves a computationally tractable solution to the image interpretation problem. Models associated with objects that are expected to appear in the scene are previously recorded in the knowledge-base of the system. Features deriver from image are matched against selected attributes of object models utilizing a variety of pattern recognition schemes.

2.2 A General Schematic Structure of Image Processing

Image processing schemes depend upon the purpose. A general schematic structure of image processing is shown in Fig.2.2 [49].

2.2.1 Picture Processing

This "processing box" (see Fig.2.2) is low-level processing and mainly bases on signal level processing and has the following characteristics.

(1) Mostly local parallel operations;
(2) Homogenous processing of all pixels;
(3) Use of special non-computer devices (e.g. optical processing);
(4) Use of pictorial input/output devices;
(5) The main problems in computer processing include:
  (a) handling of huge amounts of homogenous data with inherently two-dimensional structure;
  (b) effective packing of information in memory;
  (c) effective implementation of parallel operations.

2.2.2 Numerical Processing

This "processing box" (see Fig.2.2) is intermediate level processing and contains both signal level processing and symbolic level processing and has the features as follows.

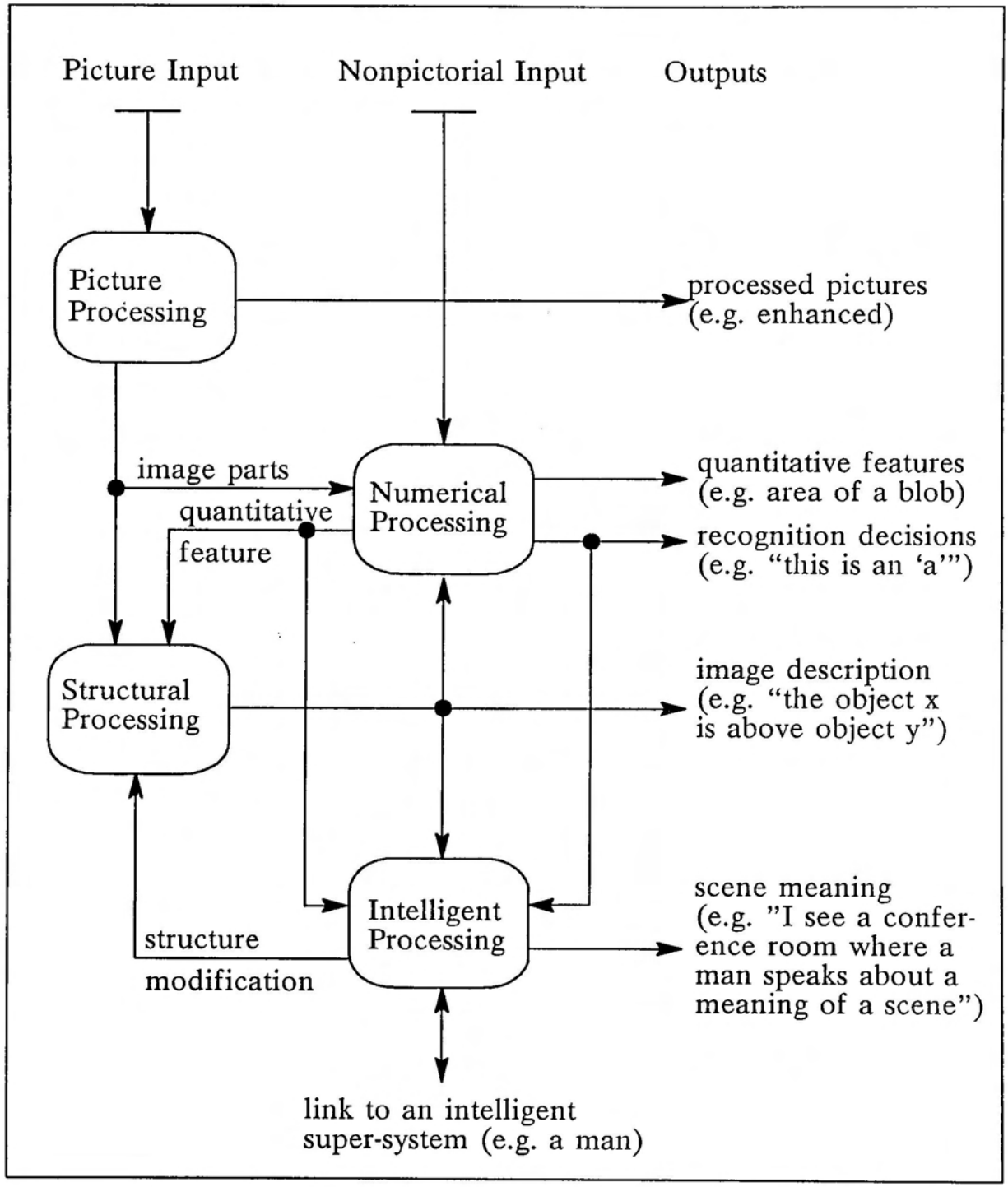

Fig.2.2. A general schematic image processing structure.

(1) Arithmetic;

(2) Simple logical (Boolean) operations;

(3) Comparatively small quantities of data, practically without structure (numerical and logical variables);
(4) No specific problems with operation repertoire;
(5) Covered excellently by ordinary general-purpose high-level languages (e.g. PASCAL, FORTRAN, C).

2.2.3 Structural Processing

This "processing box" (see Fig.2.2) is also intermediate level processing includes both signal level processing and symbolic level processing and is characterized as follows.

(1) Highly entangled data structures (lists);
(2) Rather large amounts of data, mostly of structural character (references or links to data items);
(3) Basic operations: access to structure elements (link following);
(4) Problem of choice of basic list element and its computer representation;
(5) Sophisticated memory management system needed.

2.2.4 Intelligent Processing

This "processing box" (see Fig.2.2) is high-level processing and performs only the symbolic level processing and is featured as follows.

(1) Data interpretation based on stored knowledge;
(2) Knowledge-based inference and reasoning;
(3) Various artificial intelligence devices;
(4) Based mainly on structural processing.

2.3 Taxonomy of Image Processing Machines

Over last twenty years, the semi-conductor technology has been developing by leaps and bounds. Large-scale integrated circuit technology is now making a powerful impact on the design of computers, especially, the flexibility and cheapness of commercially obtainable microprocessors have brought the chance of configuring a computer for the special job in hand within the range of quite modest research budgets. This led to the emergence of many parallel machines to solve particular problems [1, 4, 7, 10, 13, 19, 28 29 etc.]. The classification of these special purpose parallel machines is very significant for studying the parallel processing and computer architectures.

The best known taxonomy for parallel computers, proposed by Flynn [23, 24], is based on the multiplicity of the instruction and data stream, which identifies four classes of computer: 1) Single Instruction stream Single Data stream (SISD) computers corre-

spond to regular Von Neumann computers that can execute only one instruction at a time and process only one datum at a time; 2) Single Instruction stream Multiple Data stream (SIMD) computers are based on a central program controller, that drives the program flow, and on a set of Processing Elements (PE), that receive and execute the same instruction broadcasted by the central controller but elaborates their own data. The PE's consist of regular Arithmetic Logic Units (ALU) with a certain amount of local storage (registers and memory); 3) Multiple Instruction stream Single data stream (MISD) computers are very often associated to pipeline computers, where a sequence of processors elaborates a stream of data which flow through it. Each processor performs some processing on its input data stream and produces an output data stream for the next processor in the sequence. MISD pipeline computers are used to process images in raster scan mode. The pixels extracted at a time from the image feed the pipeline and go through different operations at each stage; 4) Multiple Instruction stream Multiple Data stream (MIMD) computers are made of several regular Von Neumann computers, each of which executes its own program, operating on private data. Usually a common memory is provided for communication, but other techniques can be utilized depending on the number of processors, the expected access to the shared data base and the interconnection networks.

The multiplicity of the instruction and data stream leads to very rough classification of parallel computers, which turns out to be inadequate to cover the large amount of existing implementations. Many of so called SIMD architectures depart from the strict Flynn's definition, so that there is really a continuum from "pure SIMD" to "pure MIMD" rather than a simple dichotomy. Thereby, Skillicorn extended Flynn's classification of architectures to be more discriminating, (in particular, the growing variety of multiprocessors can be categorized and related) [84], Maresca, Lavin and Li proposed the taxonomy by using three parameters, the autonomy, the network topology and the data width, for the refinement [61].

On the other hand, the parallel computers can be also categorized according machine construction and its processing morphology [42, 45, 46]. They are: 1) fully parallel processor; 2) local parallel processor; 3) pipeline processor; and 4) multiprocessor, as specifically presented in following.

2.3.1 Fully Parallel Processor

This uses the arrayed processor elements (PE's) to match two-dimensional image data as illustrated in Fig.2.3. Each PE deals with only one pixel so that this kind of system operates in parallel at pixel-level. The system usually employs thousands of or tens of thousands of homogenous PE's. Representative systems are CLIP4, developed at the University College of London, with Duff as the chief investigator [28], DAP, developed at

International Computer Ltd. of Stevenage, England, with Reddaway as the principle investigator [37], MPP, developed at Goodyear, with Batcher as the principal investigator [8, 9] AAP, developed at Nippon Telegraph and Telephone Public Corporation, Japan, with Kondo as the principle investigator [48], CM-2, developed at Thinking Machines Corp., with Tucker as the principal investigator [95, 99]. Table 2 shows the comparison of these four fully parallel machines.

Table 2.1
Comparison of Typical Fully Parallel Machines

Machine	Number of PE's	PE's/chip	Maximum Execution Speed	Local Memory
CLIP4	96×96	8	46 MIPS (Binary pointwise logic or 8-bit pointwise add)	32 bits
DAP	64×64	16	10240 MOPS (Boolean operations)	4 kilobits
MPP	128×128	8	6553 MOPS (Addition)	1 kilobits
AAP-2	256×256	64	10000 MIPS (Boolean operations)	1 megabits
CM-2	(128×128)×4	16	6000 MIPS (Document search)	64 kilobits

This kind of system performs fully parallel processing at each pixel, and acts at high-speed. But it employs too many PE's, and the interconnections between PE's are fixed and large-scale. Moreover, the image size processed at a time is limited by the number of arrayed PE's. And the programming becomes difficult when it executes processing in which parallelism does not exist, such as the calculations of geometric area and histogram.

2.3.2 Local Parallel Processor

This is the minicomputer based image processing system in which the most often used time consuming image processing operation is implemented by hardware that is added to image memory. Examples of this kind system are: TOSPIX [44] and PPP [43], developed at Toshiba Corporation, Japan, with Kidode as the principal investigator, DIP [32, 33], developed at Delft University of Technology, Delft, The Netherlands, with Gerritsen as the principal investigator, MFIP [88], developed at Osaka University, Japan, with

Sugimoto as the principal investigator.

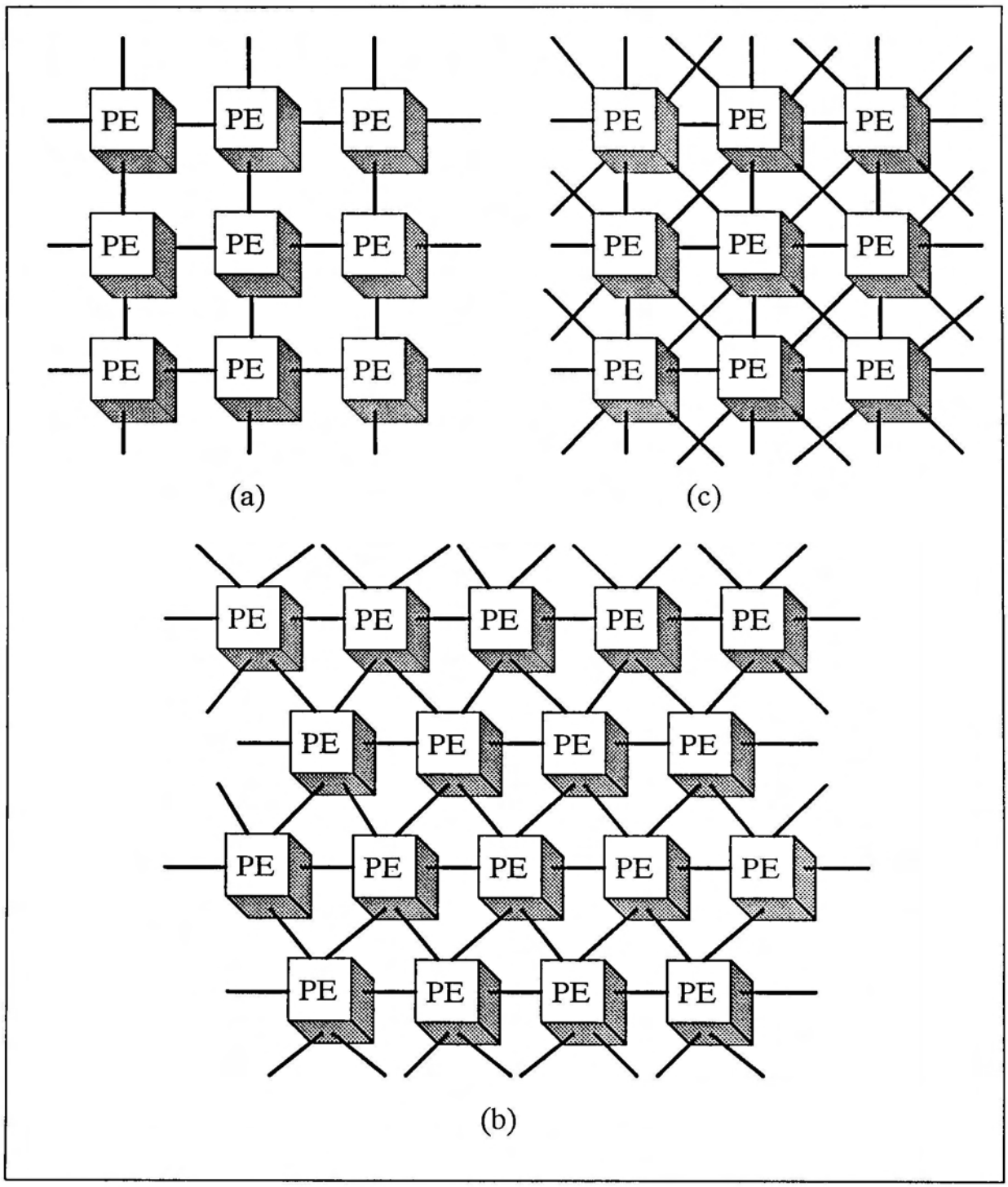

Fig.2.3. Typical structures of fully parallel machines. (a) Cross connectivity; (b) Hexagonal connectivity; (c) Square connectivity.

This kind of system can be realized easily and economically by developing the image memory module and image processing module independently, and can deal with larger size image data in comparison with the fully parallel processor. Whereas the processing speed is inferior to the previous.

2.3.3 Pipeline Processor

This kind of system makes use of PE's connected in the shape of pipeline and performs image processing sequentially as shown in Fig.2.4. Typical examples are: Cytocomputer [116], developed at Michigan Environment Research Center, with Loughheed as the chief investigator, FLIP [31], developed at Research Institute for Information Processing and Pattern Recognition, West Germany, with Gemmar as the chief investigator, GOP [34], developed at Linkoping University, Sweden, with Granlund as the chief investigator, RAPID [62], developed at Minolta Camera Co., Ltd., Japan, with Masaki as the chief investigator, TIP [29, 93, 94], developed at C & C Systems Research Laboratories, NEC, Corporation, Japan, with Temma as the chief investigator, IDATEN [47, 75, 76], developed at Fujitsu Laboratory Ltd., Japan, with Sasaki as the chief investigator. Because the moving and processing of data are performed at the same mode, the construction of this kind of system is simple. But the system is less flexible and can only handle a fixed-type processing.

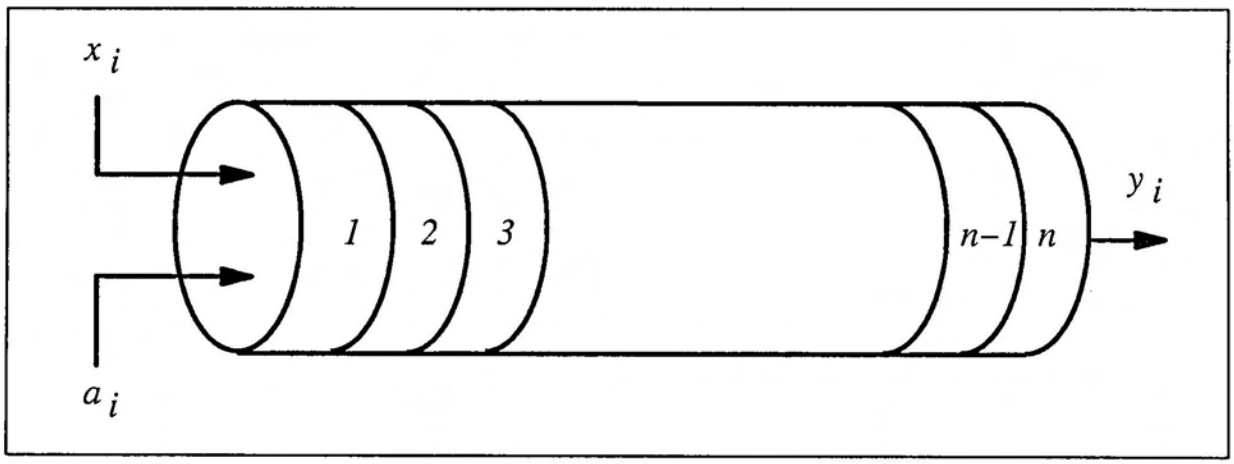

Fig.2.4. Principle of pipeline processing.

2.3.4 Multiprocessor

This kind of system utilizes multiple processors. Samples of this type are: PASM [79, 80, 81, 82], developed at Purdue University, USA, with Siegel as the chief investigator, ZMOB [51], developed at University of Maryland, USA, with Kushner as the chief investigator, MACSYM [47,116], developed at Kyoto University, Japan, with Inagaki as the chief

investigator, SIPS [36, 116], developed at SONY Corporation, Japan, with Hasebe as the chief investigator, EMMA [60], developed at Elettronica San Giorgio, Italy, with Manara as the chief investigator. Interconnections between processors are changeable upon the processing demand. Therefore, the system can be applied either to the preprocessing and high-level processing. However, to perform the real-time processing and analysis, A large number of processors is needed, and the control procedures of processors becomes complicated.

2.4 Purpose of This Research

Some of above machines can handle image processing at (or nearly to) the video-rate, e.g., SIPS can perform the filtering at video-rate, IDATEN takes only 120-ns to process a pixel, EMMA-MAORS can read the microfilm rolls at the speed of 2000,000 photograms per hour. But all of these are limited at the low-level image processing. It is hard for these machines to handle the real-time high-level image processing and understanding. Because most of them employ the arithmetic circuit as their PE, the programming becomes difficult when to perform the high-level processing.

The purpose of this research is to develop a new parallel machine to deal with the real-time both low-level and high-level image processing and analysis problem. The programming language of this new parallel machine is the commercially obtainable high-level language which has high programmability.

2.5 A Proposition of a New Computer Architecture for the Real-time High-level Image Processing and Analysis

In above, the classifications of image processing and image processing machines are discussed, and their advantages and disadvantages are related. Especially, for those weak points of pipeline architecture's and multiprocessor's, *a system whose architecture is the combination of the pipeline and multiprocessor architectures will preserve the advantages of both and remedy each other's disadvantages.* HIGIPS just utilizes this proposition. It takes the pipeline architecture in whole, and the multiprocessor for each stage. The multiprocessor used here is simply a shared bus and shared memory type, and employs the general-purpose microprocessor as its PE which also has a local memory. The specific design concepts of HIGIPS will be given in the next chapter. .

Whenever the architecture of a computer is radically different from that of the conventional Von Neuman type, the programmability of this new machine is a delicate and crucial feature for obtaining high overall performance. Moreover, the programming language for such a computer should both be independent from it (portable and readable)

and efficient on it : two truly conflicting requirements! For these reason, different ap proaches have been suggested for handling this problem, generally strongly influenced b the previous experience gained in this field by those putting forward their ideas. There i no general consensus as to whether it is better to program in a high-level language (suc as, for example Fortran, Pascal or C) and to call from an image processing library sub routines optimally designed for a given machine; or to use an interpreter like APL i which interactivity and accurate diagnostics may be obtained; or, finally, to define a high level language having specific control structures for local computations and global paral lelism, as well as data structures and data types particularly useful in this field. In shor programmability is indeed a key issue in designing a system that can be used for imag processing in an efficient yet flexible way.

HIGIPS will employ the present existent high-level programming languages such a Fortran, C and so on, which are certainly independent from HIGIPS, and running o personal computers. The software strategy of HIGIPS will be discussed in chapter 6.

Chapter 3

DESIGN CONCEPTS OF HIGIPS MACHINE

On the basis of the analysis of conventional image processing machines, this chapte discuss the design of HIGIPS. As known, at the present day, the advanced semi-conducto technology and large-scale integrated circuit technology make it possible that a 1 mm^2 chip can include hundreds of or thousands of thousands transistors. This brings the ap pearance of extra large-scale integrated circuits, for example, 16 megabits DRAM IC's 32- or 64-bit general-purpose microprocessor IC's, and reduces the costs of these IC's.

Along with the vigorous development of semi-conductor technology and large-scal integrated circuit technology, the software technology also made a remarkable progress after breaking through the software crisis in 1960's. The programming environment ha been improving in the following sides.

(1) Various of high-level languages, for instance, Fortran, C, Prolog, have been developed;
(2) Many syntax oriented editors easily to handle have been elaborated;
(3) The programs can be reused by making them into library routines;
(4) Conversational debugging tools have been built on.

Under this situation of the hardware and software technology, i.e., the hardware an software resources are quite abundant and obtainable economically, the people can't hel think: can we make a high-performance machine with use of these hardware and softwar resources? The HIGIPS is its answer which is conceived for parallel image processing understanding and analysis.

3.1 Block Diagram of HIGIPS

The HIGIPS's architecture is the combination of the pipeline and multiprocessor ar chitecture, i.e., as a whole, it employs a pipeline architecture each stage of it employs th multiprocessor architecture. As illustrated in Fig.3.1, HIGIPS is composed of 1) Pictur Input Unit (PIU); 2) *N* Picture Processing Units (PPU); 3) Picture Output Unit; and 4 System Controller.

3.1.1 Picture Input Unit

This unit comprises the A/D converter module (ADM), memory module (MM), an PIU controller (PIC). ADM digitizes the analogous video signal from the video camera

VTR or other devices, and sends the digitized image data into MM under the control of PIC.

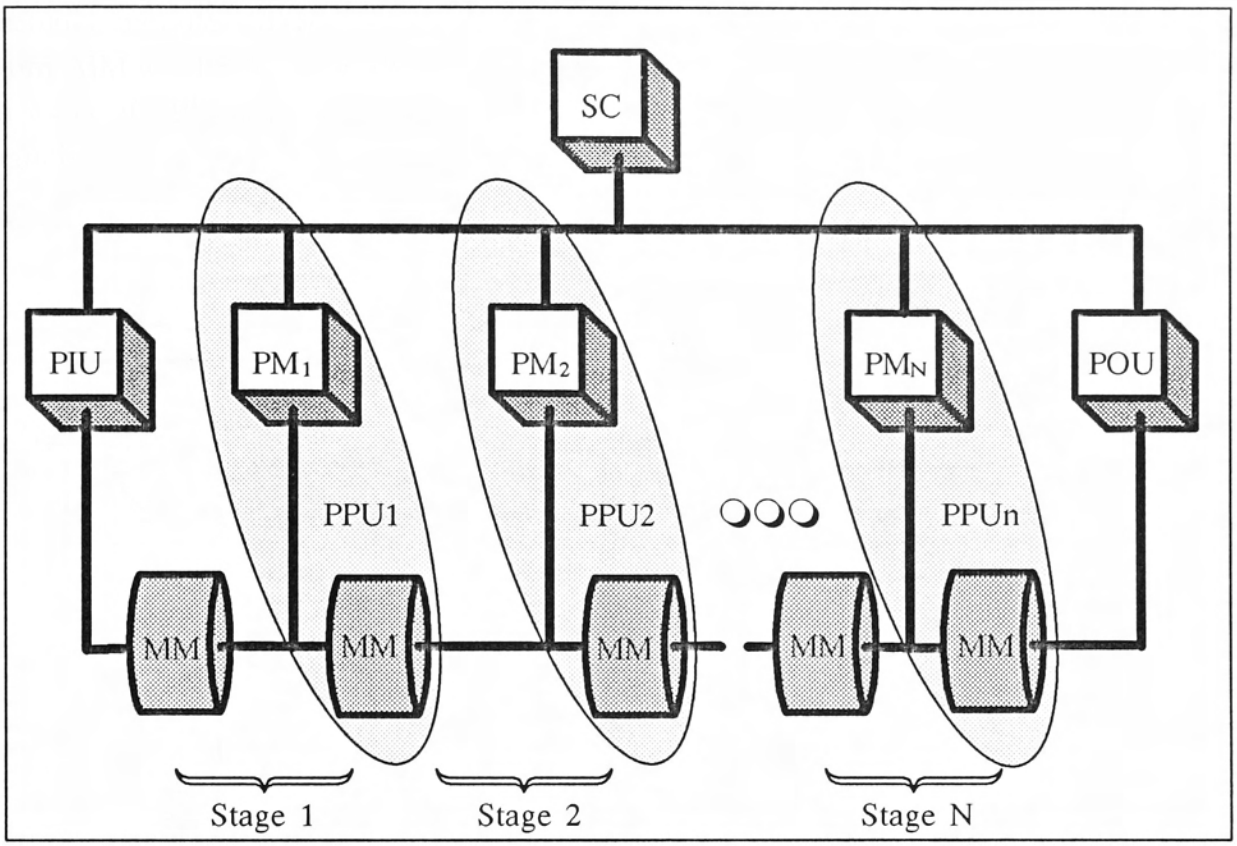

Fig.3.1. Block diagram of HIGIPS.

3.1.2 Picture Processing Unit

The components of PPU are the processor module (PM) and MM that is a time-sharing dual port memory, and plays the role of the global memory (GM) of the stage (See Fig.3.2). PM is composed of the common ROM (CROM) and *M* processors that have the local memory (LM). One of them functions as sub-controller (SUBC), the others are slave processors (SMPU). SUBC is responsible for downloading the processing program from SC into GM, transferring the downloaded program into each LM of SMPU, starting to run the programs in LM's, monitoring the activity status of each SMPU, sending the end message of each SMPU to SC. And it is managed by SC.

MM is constituted by two blocks of large capable SRAM's and bus switches (BS and BS'), and works as the dual port memory. The PM is connected to the memory block $M_{i,1}$ and $M_{i+1,2}$ by setting BS *on* and BS' *off*, and vice versa, to memory block $M_{i,2}$ and $M_{i+1,1}$.

3.1.3 Picture Output Unit

This unit consists of the frame buffer memory module (FBM), POU controller (POC), memory module (MM), and VTR or display. The final processing result is transferred from MM to the VTR or display via the FBM under the control of POC. The buffer memory of FBM must be large enough to output not only black-white but also color video signals.

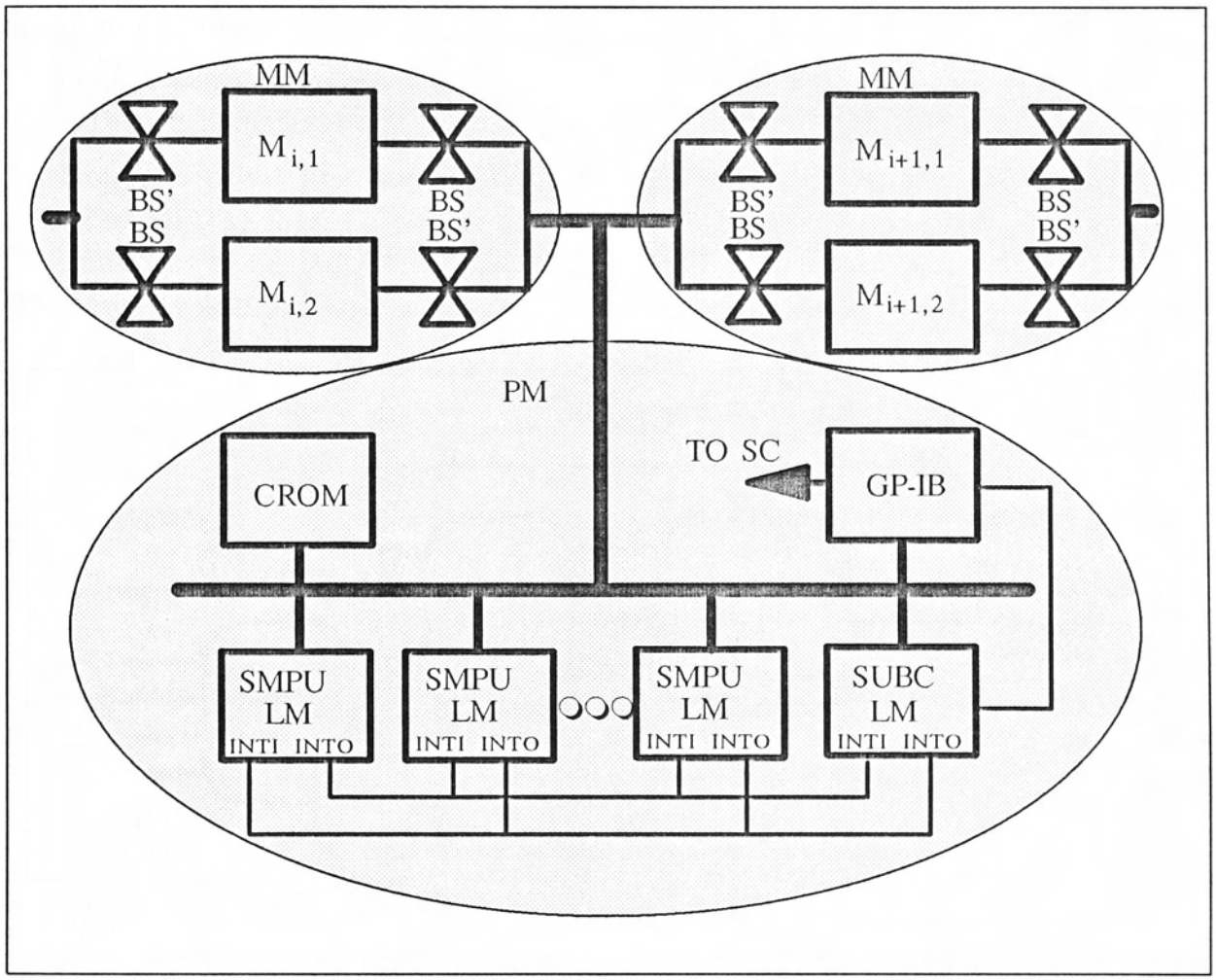

Fig.3.2. Picture processing unit.

3.1.4 System Controller

SC is a personal computer or minicomputer that takes the control of the whole system. Its functions are to develop the processing program with high-level languages (e.g., C, FORTRAN), to download the executable program into GM of each stage, to monitor the activities of each stage, to set BS and BS' *on/off*.

3.2 Design of PM

As shown in Fig.3.2, the PM consists of multiple PE's. The efficiency of the system is based on the performance of PE. Therefore, the design and the development of PE is the key issue of a new system aiming at high-speed and high performance.

3.2.1 Processing Element

There are two kinds of PE's utilized in the traditional systems specialized for parallel image processing.

(1) The general-purpose microprocessor;

(2) The arithmetic circuit such as adder-subtracter, multiplier-divider etc..

Examples of the first type are PASM (16 MC68000 MPU's) [79, 80, 81, 82], ZMOB (256 Z80 MPU's) [51], MACSYM (16 Z8000 MPU's) [47, 116], PX-1 (32 Z80 MPU's) [77]. Samples of the second type are TIP-1 [29, 93, 94], SPARC[46], DIP [32, 33], SIPS [36, 116]. Typical structure of this type is illustrated in Fig.3.3.

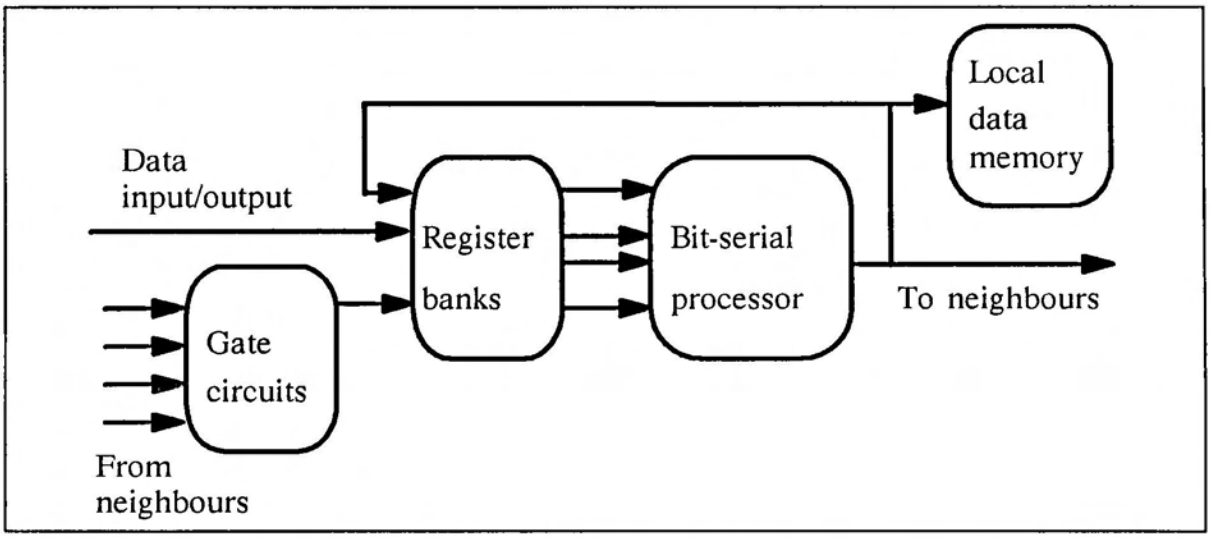

Fig.3.3. Typical structure of PE based on the arithmetic unit.

Although it is becoming easily to design and order the custom IC's, yet it is high-cost and time-consuming for ordinary users. And also, the special designed PE is less flexible and is hard for high-level processing. In comparison with these weak points of the second type PE, the general-purpose microprocessor is low-cost and is good at high-level processing. Moreover, the software resources of NEC PC98 series personal computers are abundant and are well built. Therefore, the general-purpose 16-bit microprocessor NEC V50 (μPD70216) is chosen as the PE of HIGIPS, which is compatible with the present software resources of NEC PC98 series personal computers..

3.2.2 Accesses of Image Data

The followings are the most often used methods to access the image data.

(1) Raster scan;
(2) Common memory;
(3) Distributed memories;
(4) Combination of a common memory and distributed memories.

The raster scan principle is shown in Fig.3.4. It can be implemented easily, but the accessing is fixed. Its representatives are FILP [31], DIP [32, 33].

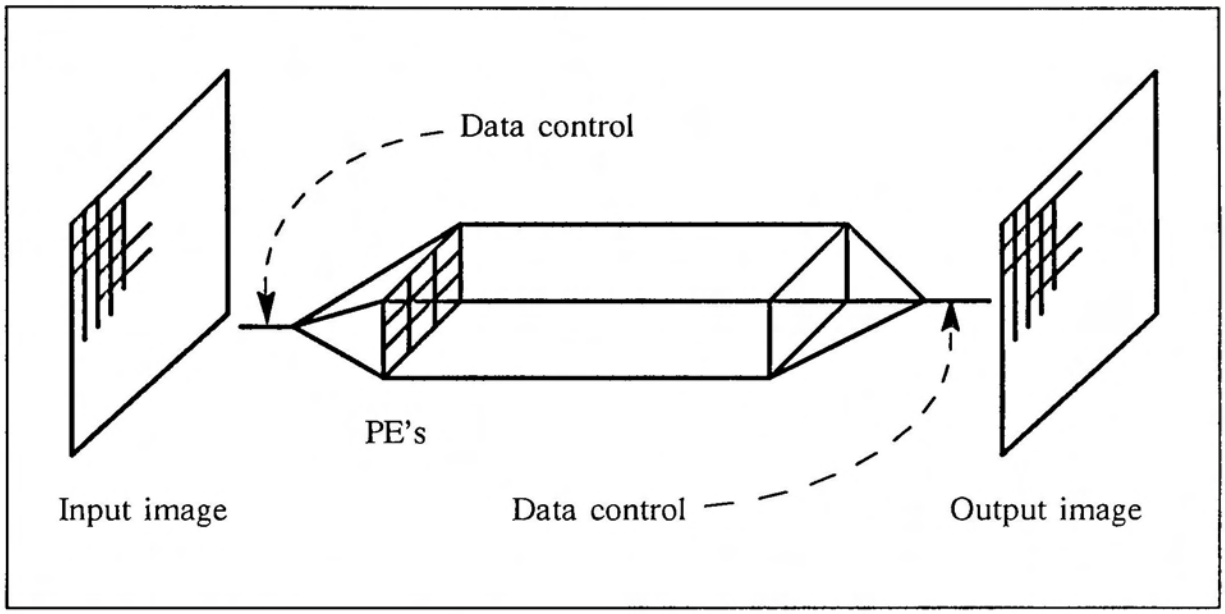

Fig.3.4. Principle of raster scan type image processing.

Fig.3.5 is the common memory scheme. All PE's can access the image memory randomly so that the efficiency of the memory is wonderfully increased. But the input/output control of image data becomes quite difficult accompanying with the increase of the number of PE's. TIP [29, 93, 94], SPARC [46], SIPS [36, 116] are systems using this access scheme.

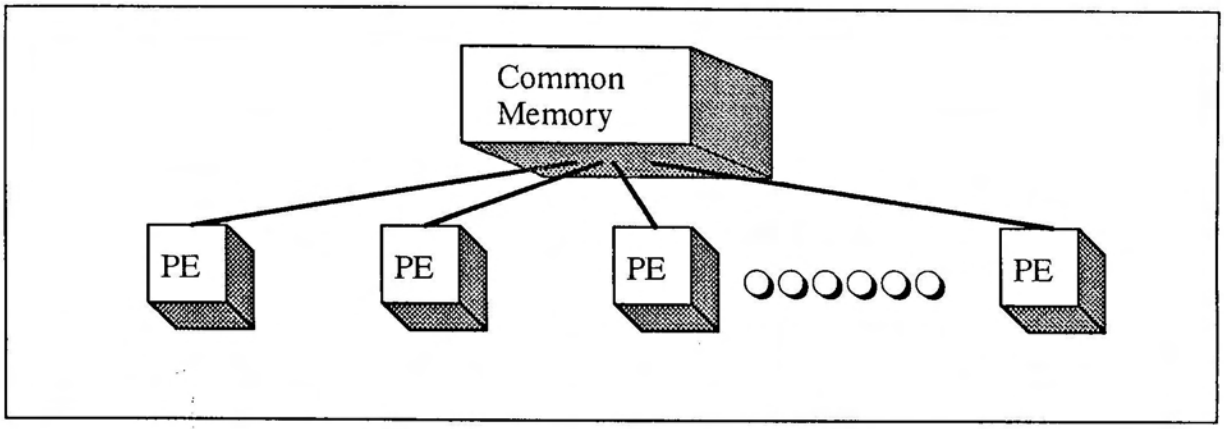

Fig.3.5. Common scheme of machines with common memory.

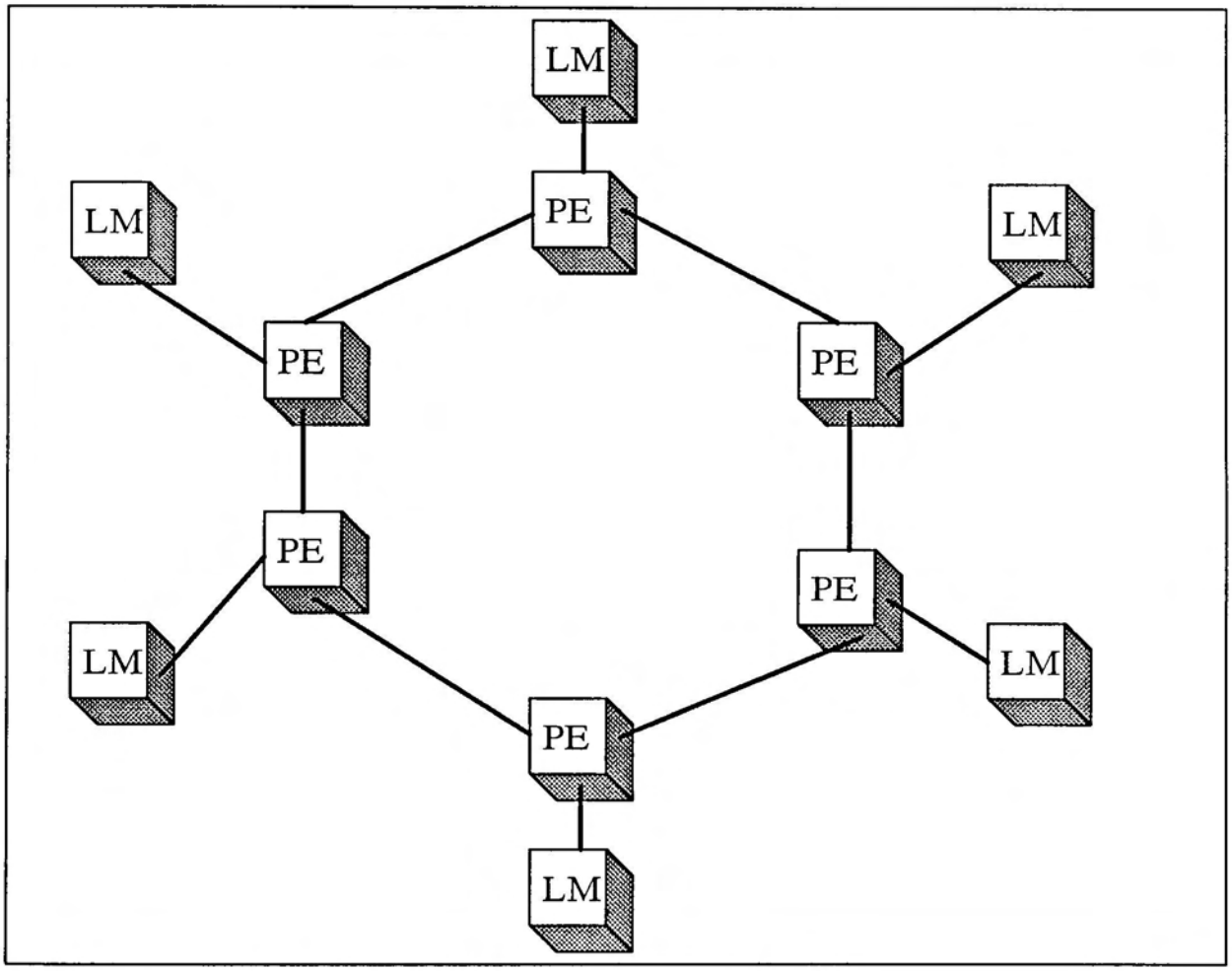

Fig.3.6. Scheme of systems with distributed memories.

The system with distributed memories is schematized in Fig.3.6. This kind of system can perform low-level image processing at high-speed because each PE deals with only its

own image data. But when to process large-scale image, the communications between PE's, i.e., the message exchanges and image data exchanges, become difficult and complicated. Typical example of this kind of system is ZMOB [51].

For the disadvantages of these three image data accessing schemes, PASM [79, 80, 81, 82], MACSYM [47,116] and PX-1 [77] etc. used the combination of the common memory scheme and distributed memories scheme to remedy each other's disadvantages and increase the memory efficiency.

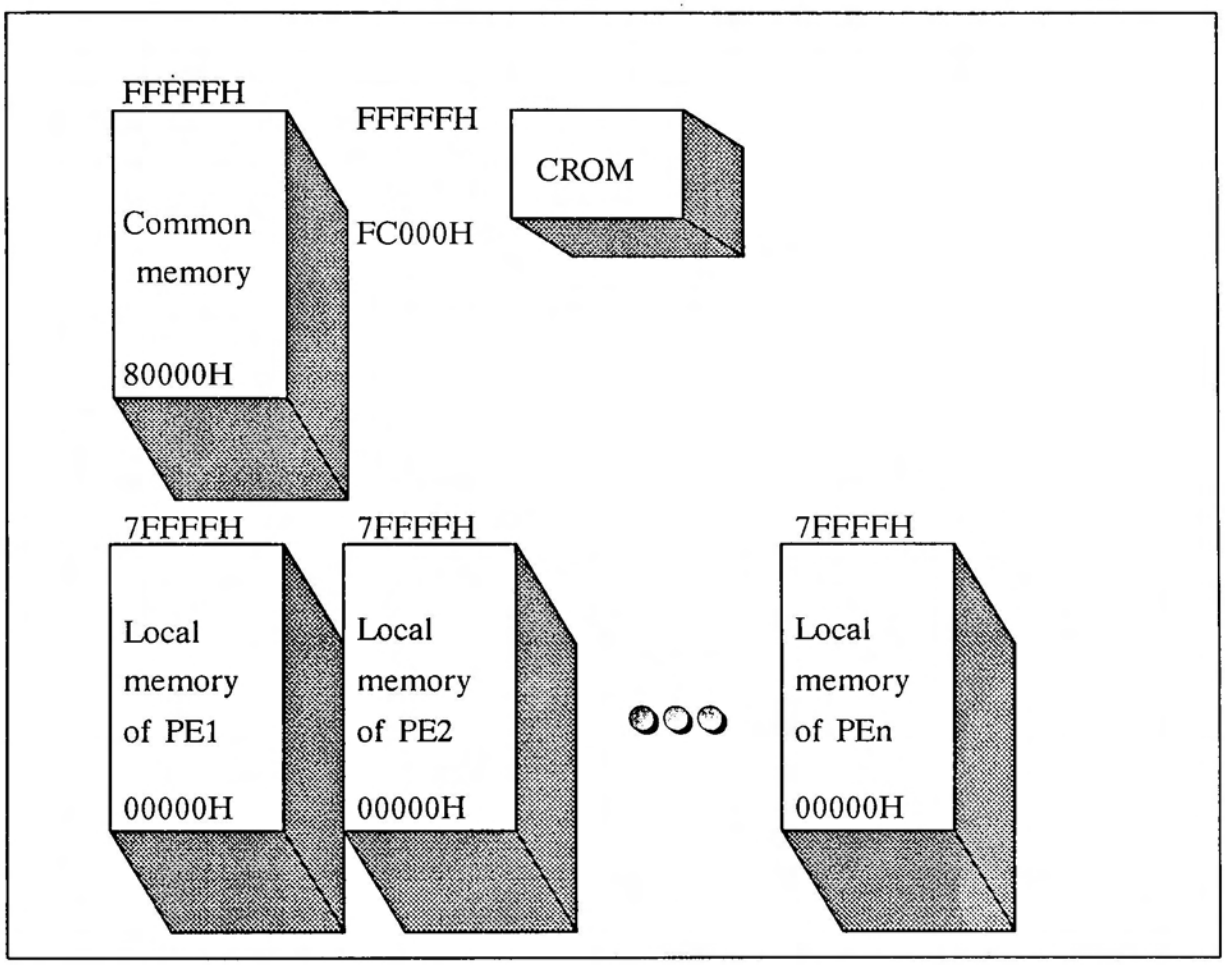

Fig.3.7. Memory map of HIGIPS.

HIGIPS is also adopted this combination to improve the memory efficiency. The PE of HIGIPS is NEC V50 microprocessor and has 1-megabyte memory space. This 1-megabyte memory space can be divided into upper memory block (UMB) and lower memory block (LMB). UBM is used common memory for image data and LMB is used as local memory for private image data and processing program. Both memory blocks can be up to 512 kilobytes. Fig.3.7 illustrates one stage's memory map of HIGIPS. The top 16 kilo-

bytes are superimposed with common ROM (CROM) area. The CROM keeps the program for starting up the HIGIPS. When HIGIPS gets into the steady state, CROM is disconnected automatically. UMB is fully used for image data.

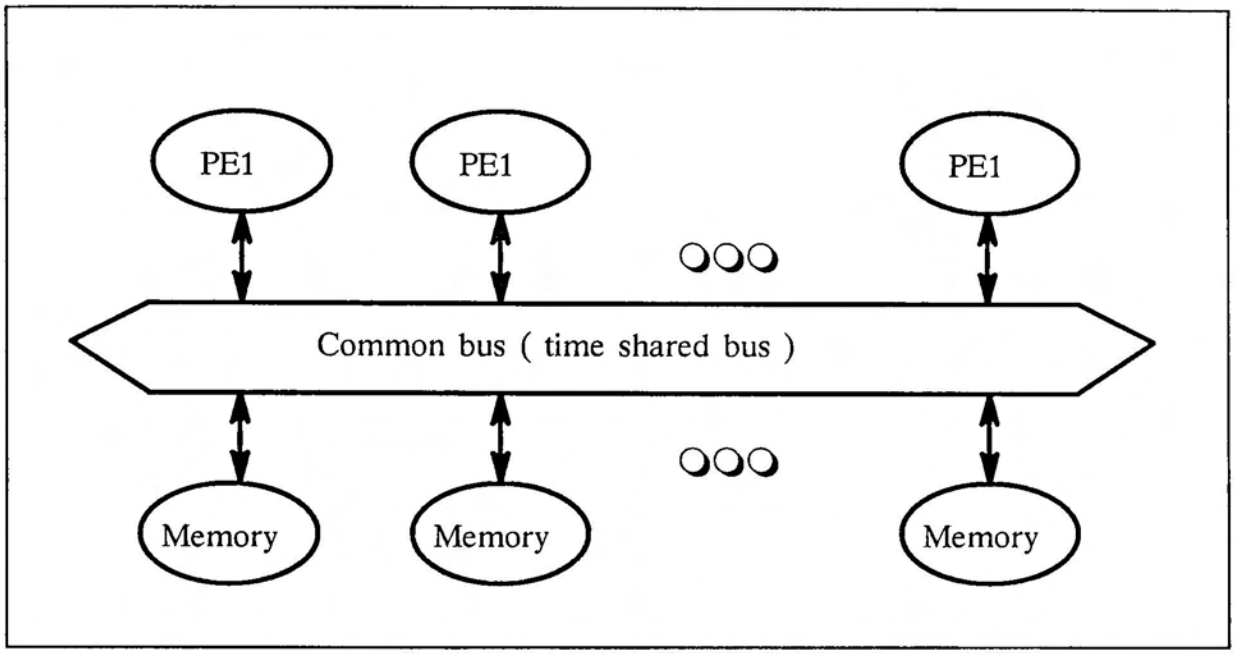

Fig.3.8. Scheme of common bus systems.

3.2.3 Message Exchange

The followings are the methods used for message exchange in traditional systems.

(1) Common bus;
(2) Ring bus;
(3) Crossbar switch circuit;
(4) Cube network;
(5) Combination of common bus and ring bus;
(6) Combination of cube network and ring bus.

Examples of common bus type system are FLIP [31], DIP [32, 33], MACSYM [47, 116]. As shown in Fig.3.8, all PE's access image memories via a common bus at time-shared mode. This kind of scheme can be constructed simply at low-cost. But the data paths are fixed.

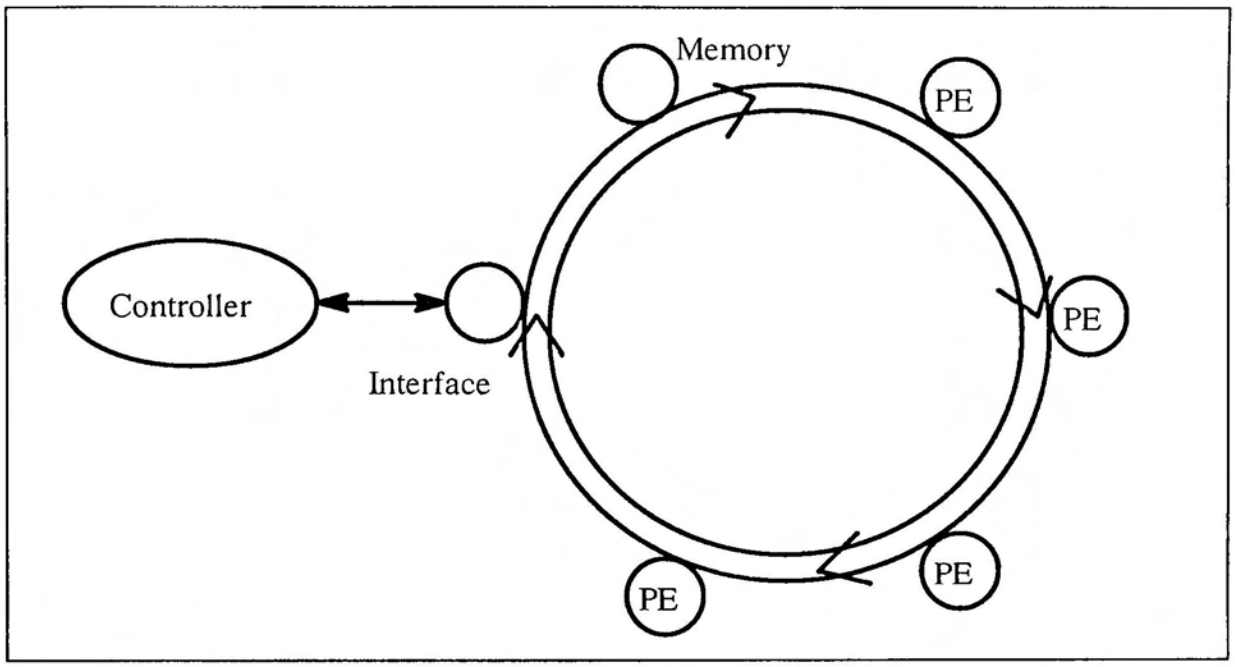

Fig.3.9. Scheme of systems with ring bus.

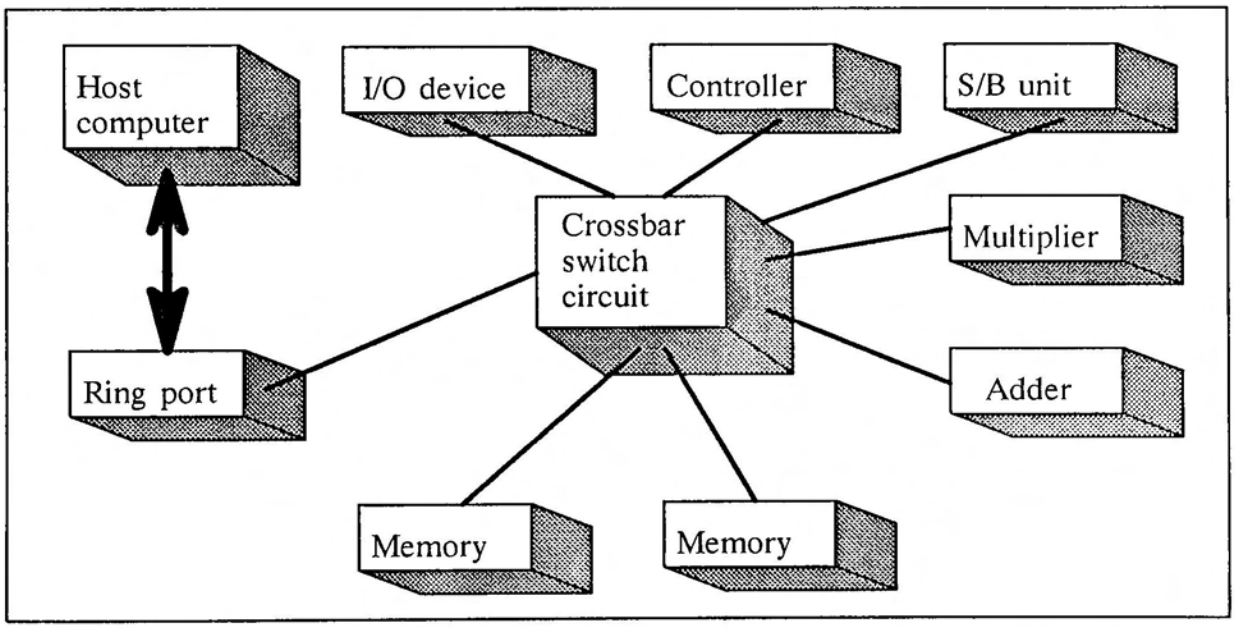

Fig.3.10. System scheme using the crossbar switch circuit for setting the data paths between PE's.

The principle of the ring bus system's diagram is the same with the common bus system as illustrated in Fig.3.9, except that the bus forms a loop. This scheme is easily to

be constructed economically. But the data path is fixed at one direction, and the system is less flexible. An example of this kind of system is TIP [29, 93, 94].

An example of the systems employing the crossbar switch circuit to set the data paths is SPARC [46] (see Fig.3.10). One of the modules, such as an adder, a multiplier, a memory etc. can connect to another randomly and all modules run in parallel. But once the data paths are assigned, they are settled definitely.

The disadvantages of the common bus, ring bus and crossbar switch circuit, lead people to think: can we make the connections between PE's flexible? The cube network is the answer. Fig.3.11.(a) illustrates the scheme of the system using the cube network as the mechanism to decide the data paths, and Fig.3.11.(b) gives the basic elements of the cube network. An example of this kind of system is PASM [79, 80, 81, 82]. Using the cube network, the data paths between PE's can be chosen as desired. This certainly increases the performance of the entire system, and brings the flexibilities to entire system. However, when the number of PE's is enormous, the network becomes large-scale and complicated, and yet is hard to be manufactured.

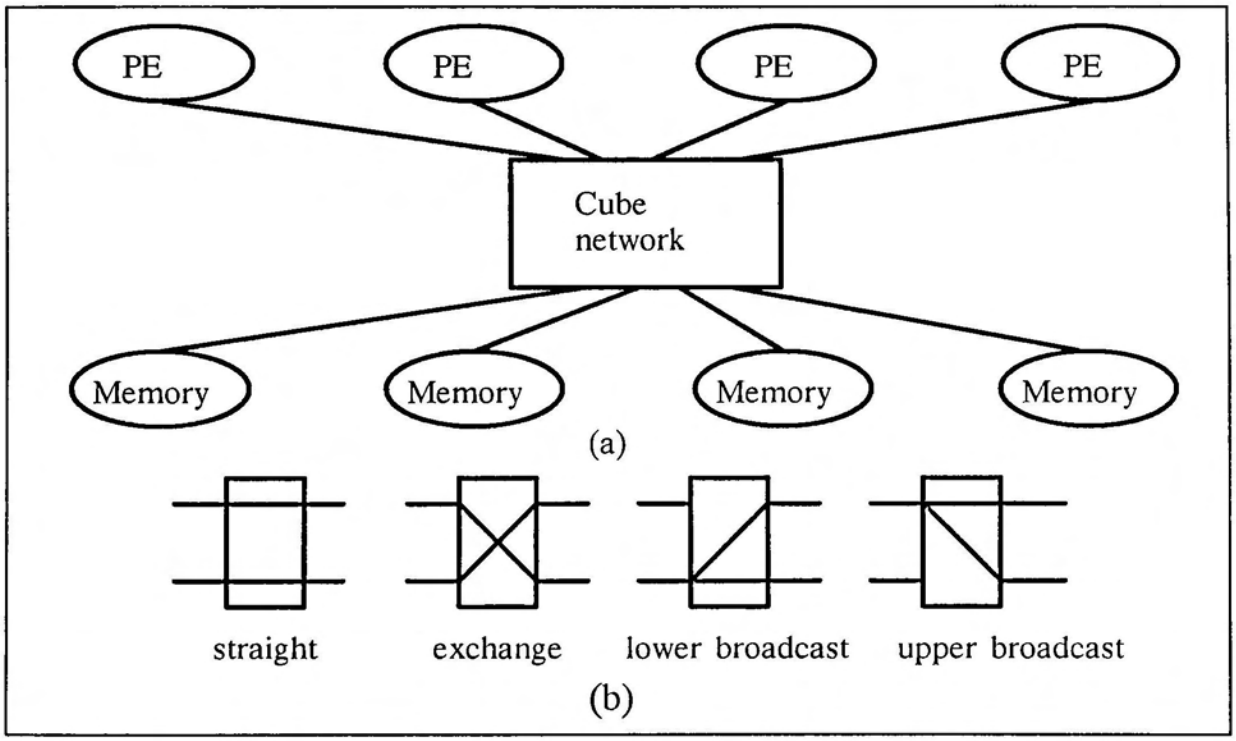

Fig.3.11. (a) System scheme with the cube network for setting the data paths between PE and memory or PE and PE; (b) Construction elements of cube network.

To shorten the developing time and decrease the cost, HIGIPS employs the most easy one—common bus type. Fig.3.12 shows the schematic drawing of PM of HIGIPS. Each PE has 512 kilobytes private memory to loose the overhead of the common memory. This can be explained as follows under the supposition that image data have already memorized in the common memory. Firstly, each PE loads the assigned partial image data into its own local memory, and then performs image processing independently, finally, sends the results back to the common memory. Each PE will apply for the use of the common bus only when it accesses the common memory. This greatly decreases the access frequency to the common memory. Hence, the overhead of the common memory is reduced. Note here that the priority of PE to access the common memory is arbitrated by the PE's bus arbiter.

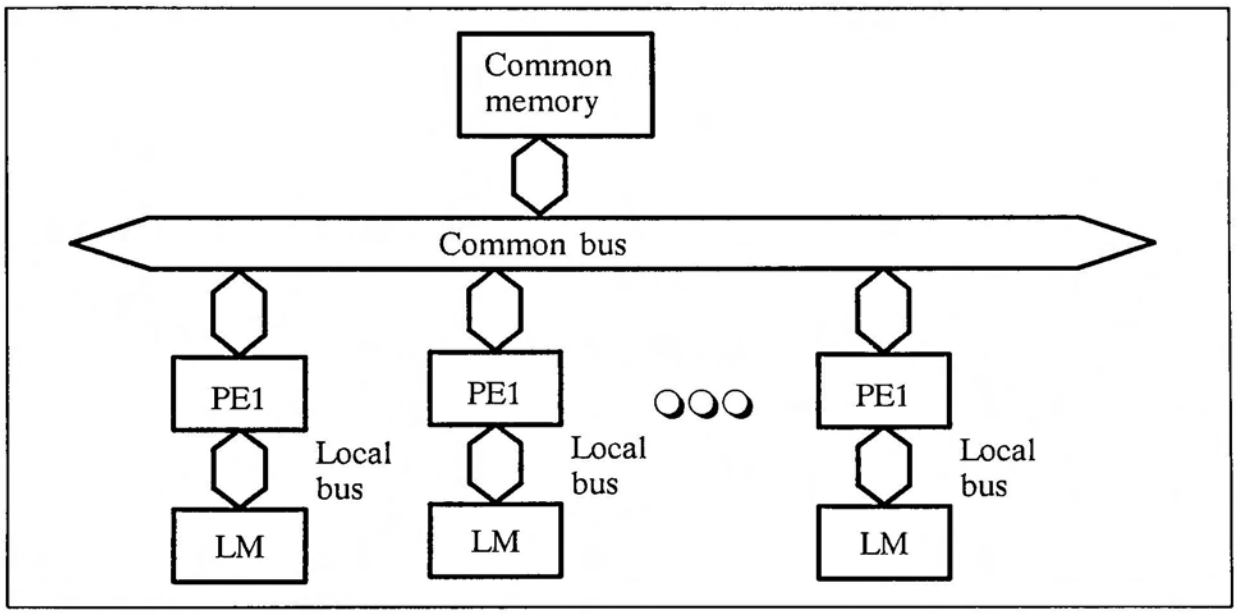

Fig.3.12. PM structure of HIGIPS.

3.3 Design of MM

The MM is used as the common memory (global memory). As mentioned before, the HIGIPS performs the pipeline processing in whole, and multiprocessor processing in each stage of it. The pipeline operation of HIGIPS is realized by this specially designed MM.

Usually, to make a pipeline computer, the dual port memory are optimal. However, at present day, the largest capacity dual port memory IC's are 2 kilobytes. To construct 512 kilobytes common memory, 256 chips of this kind are necessary. It will take a huge space. Consequently, MM of HIGIPS will not be built with these dual port memory IC chips.

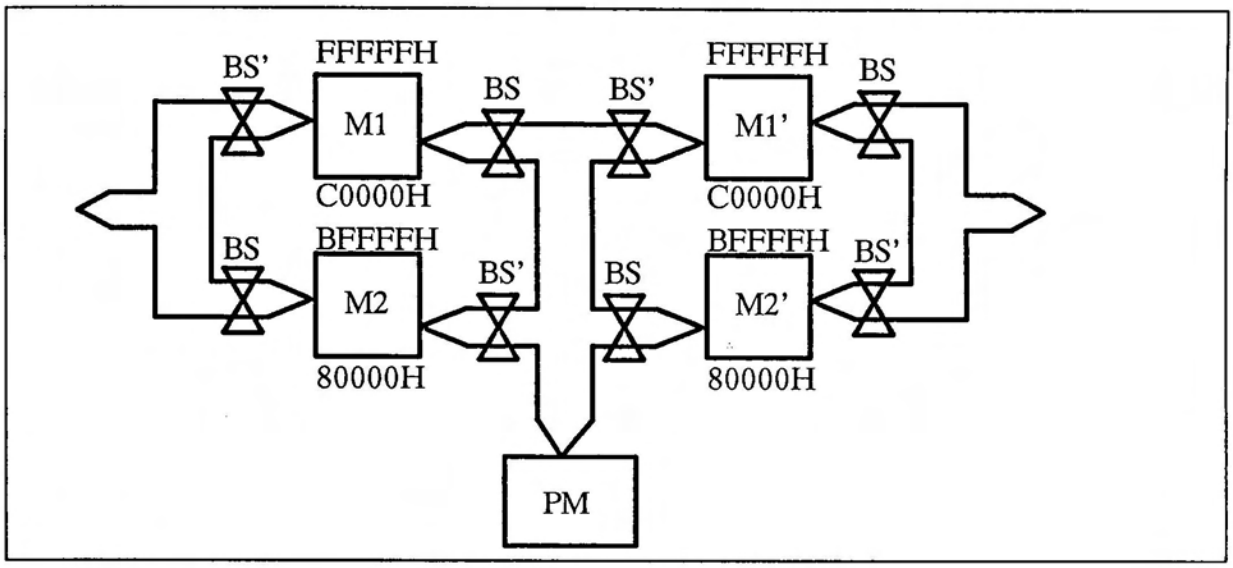

Fig.3.13. Block diagram of MM.

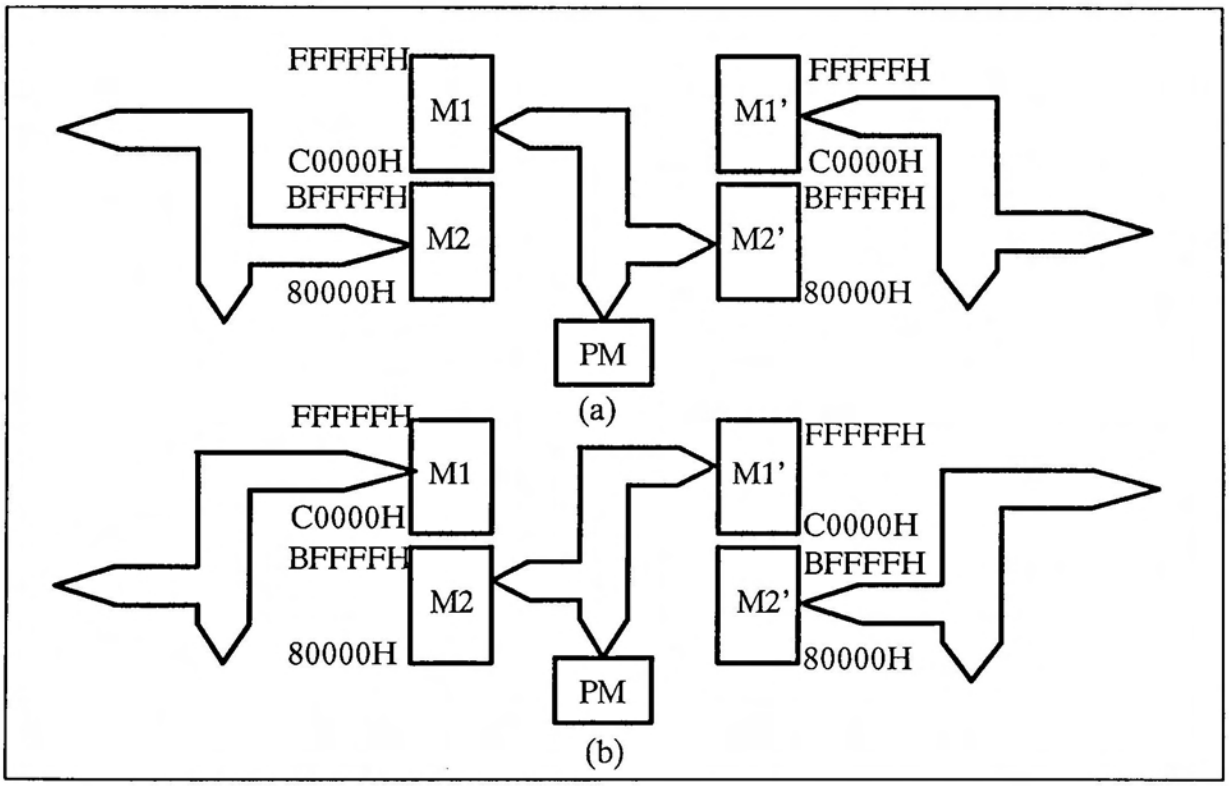

Fig.3.14. Two phases of MM: (a) Phase A by setting BS on and BS' off; (b) Phase B by setting BS off and BS' on.

MM of HIGIPS is constructed with the large capacity SRAM IC chips and some necessary buffer and latch IC chips. In the following, these buffers and latches are generally

called bus switches (BS and BS'). Each MM consists of two memory blocks M1 (from C0000H to FFFFFH) and M2 (from 80000H to BFFFFH) and two kinds of bus switches BS and BS'. M1 and M2 are both 256 kilobytes, as presented in Fig.3.13.

MM can be accessed in two phases. In phase **A**, BS is set *on* and BS' *off*. PE's of PM can access memory block M1 and M2' (see Fig.3.14.(a)). In phase **B**, BS is set *off* and BS' *on*, and the PE's can access the memory block M2 and M1' (see Fig.3.14.(b)). The pipeline operation of HIGIPS is obtained by arranging multiple MM's in a line and setting them in phase **A** or phase **B** alternatively.

3.4 Pipeline Performance of HIGIPS

The procedure of the image processing on HIGIPS is, firstly, to develop the processing program on SC with high-level languages and make the executable file, secondly, to download this executable program into local memory via GM of each stage of the HIGIPS, and finally, to run the downloaded program from local memory. But how does HIGIPS work at the pipeline mode? When the power is on, all processors (SUBC and SMPU's) in a stage will copy the monitor program from CROM to their own local memory, and start up from the local memory. After processors (SUBC and SMPU's) finish the corresponding initialization, SUBC is ready to receive the commands from SC, and SMPU's are ready to receive commands from SUBC. At this moment, the system becomes steady, and BS is set *on* and BS' *off*. The whole HIGIPS can be redrawn as shown in Fig.3.15.(a) in the case that the number of stages is 3. The *i*-th image data frame is recorded to M12. Note here that when the *(i–1)*-th image data frame was recorded, BS was *off* and BS' was *on*. From Fig.3.15.(a), it can be seen that HIGIPS has been divided into five disconnected parts. The camera is connected with M12 (i.e., the *i*-th image data frame will be placed into this memory block). While M11, which contains the *(i–1)*-th image data, is connected to the PM1 and will be used as the input data to the PM1. The PM1 will output the intermediate result to M22. And the PM2 uses the M21 (which preserves the output result of PM1 at time τ_i–τ_p) as its input and the intermediate output result will be put in M32. PM3 will use M31 as its input and output the final result to M42. M41 (which includes the the final result at time τ_i–τ_p) will be the input data of POU.

When time reaches τ_i+τ_p, SC set BS *off* and BS' *on* and HIGIPS can be redrawn as in Fig.3.15.(b). The *(i+1)*-th image data frame will be recorded to M11, and the process is not much different from the previously explained except that each PM will deal with the new input image data recorded at τ_i, or the new intermediate result sent from the left neighbour PM at τ_i. Therefore, the pipeline performance of HIGIPS is realized by setting BS and BS' *on* or *off* alternately. Note here that when the switch is *off*, the bus will be disconnected to get rid of the problem of the bus contention. And where, $\tau_p = max\{\tau_{in}, \tau_1,$

τ_2, τ_3, τ_{out}}, and τ_{in} and τ_{out} are the input and output time of PIU and POU, separately, and τ_1, τ_2, and τ_3 are the processing time of PM1, PM2, and PM3, accordingly. The smaller the τ_p, the higher the performance of HIGIPS is. When $\tau_{in}=\tau_1=\tau_2=\tau_3=\tau_{out}$, the HIGIPS works optimally at perfect pipeline mode.

Table 3.1 gives the connections between the processing modules or input/output units and memory banks in Phase **A** and **B**.

Table 3.1

Connections between Processing Modules or Input/output Unit and Memory Banks.

Phase A								
Modules	Memory banks							
	M11	M12	M21	M22	M31	M32	M41	M42
PIU	1	0	x	x	x	x	x	x
PM1	0	1	0	1	x	x	x	x
PM2	x	x	1	0	1	0	x	x
PM3	x	x	x	x	0	1	0	1
POU	x	x	x	x	x	x	1	0
Phase B								
Modules	Memory banks							
	M11	M12	M21	M22	M31	M32	M41	M42
PIU	0	1	x	x	x	x	x	x
PM1	1	0	1	0	x	x	x	x
PM2	x	x	0	1	0	1	x	x
PM3	x	x	x	x	1	0	1	0
POU	x	x	x	x	x	x	0	1

Note: "1" means that the corresponding memory banks and processing modules or input/output units are connected;
"0" means that the corresponding memory banks and processing modules or input/output units are disconnected.
"x" means that there does not exist connections between memory banks and processing modules or input/output nuits.

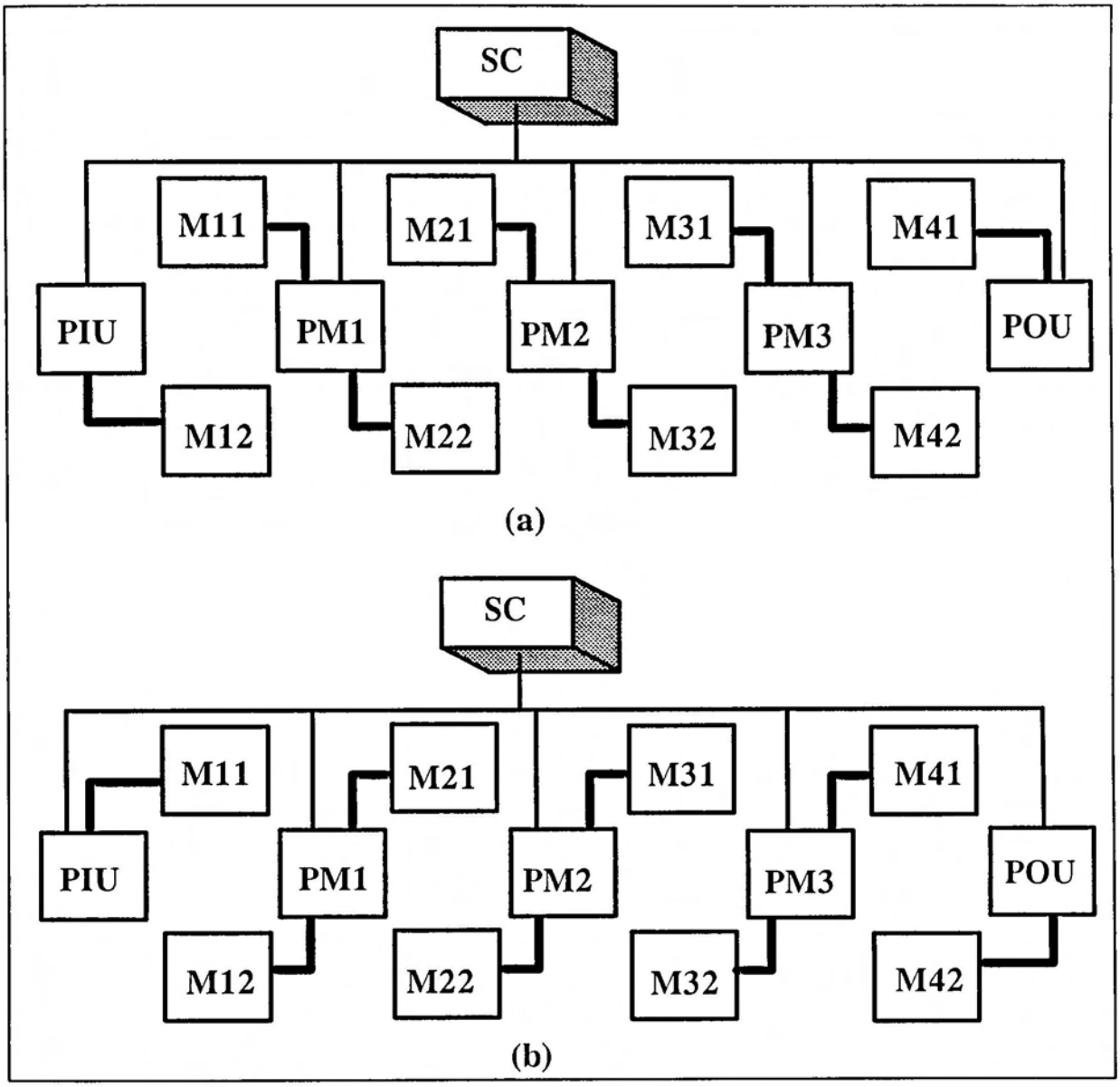

Fig.3.15. Principle of pipeline performance of HIGIPS. (a) Connections between PIU, POU, PM's and MM's at t_i; (b) Connections between PIU, POU, PM's and MM's at $\tau_i+\tau_p$.

3.5 Design of PIU

If to perform the real-time image processing, understanding and analysis, the development of image input unit is very important. It must be able to input the image data into the image memory at video rate. The block diagram of PIU is illustrated in Fig.3.16.

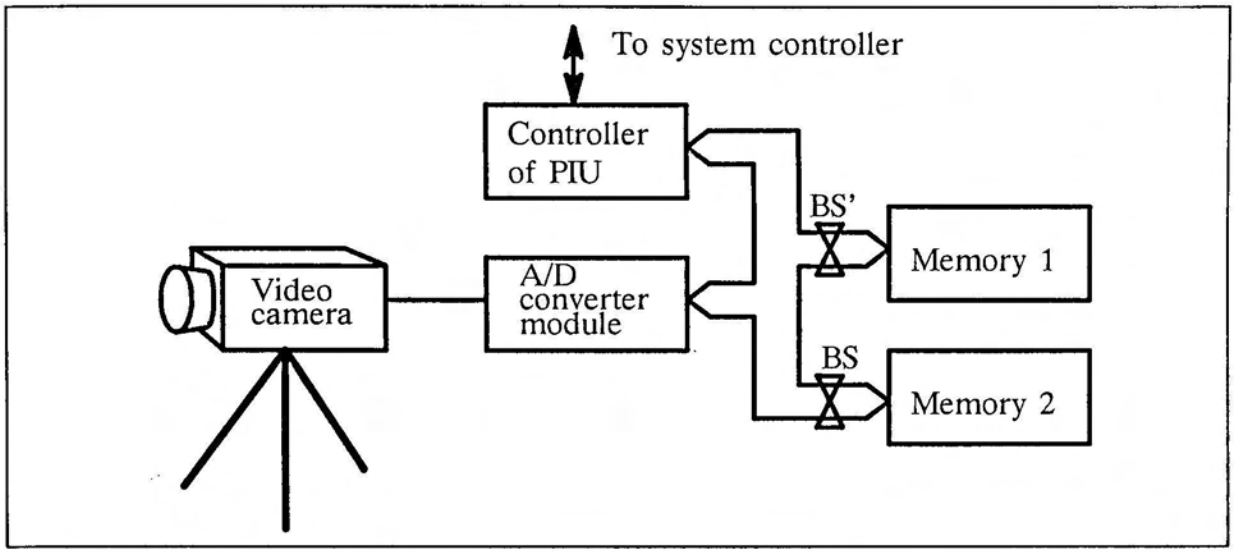

Fig.3.16. Block diagram of PIU.

The analog video signal is from the video camera. Certainly, the output of video tape recorder (VTR) and TV set can also be the input signals of PIU. The A/D converter module (ADM) digitizes the analog signal and sends the digitized image data into memory 1 or memory 2 under the control of PIC (controller of PIU). Memory 1 and memory 2 are of homogenous construction with MM discussed before. The PIC obeys the system controller.

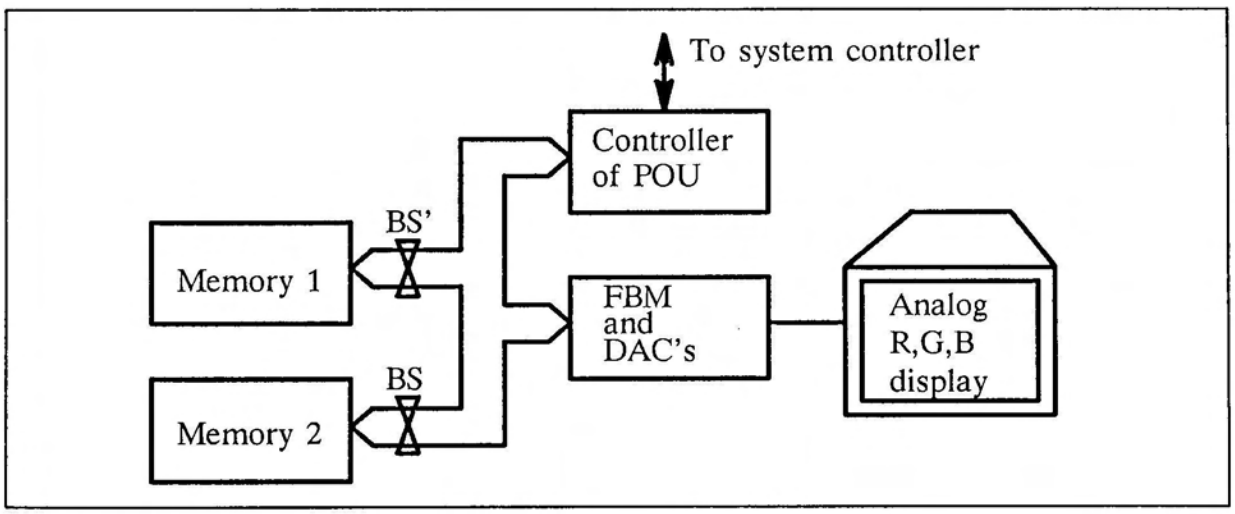

Fig.3.17. Block diagram of POU.

3.6 Design of POU

To output the processing result is another important factor of the real-time image processing. The output speed of the result affects the total performance of the system. No

matter how high the processing speed is, the system can not perform the real-time processing if the output speed can not match to the processing speed. Fig.3.17 gives the block diagram of POU.

Memory 1 and memory 2 are homogenous memory banks of MM as related before. FBM is a dual port memory. Image data can be written into it from one port, and read out from another and sent to digital/analog converters (DAC) under the control of POC (controller of POU). The output of DAC is fed to the analog RGB display.

3.7 Choice of SC

Accompanying with the cheapness of LSI's, the personal computers are getting popularization day by day. Since the most popular personal computer in Japan is NEC PC98 series products, and the software resources of it are abundant, the NEC PC98 series personal computer is selected as the system controller of HIGIPS.

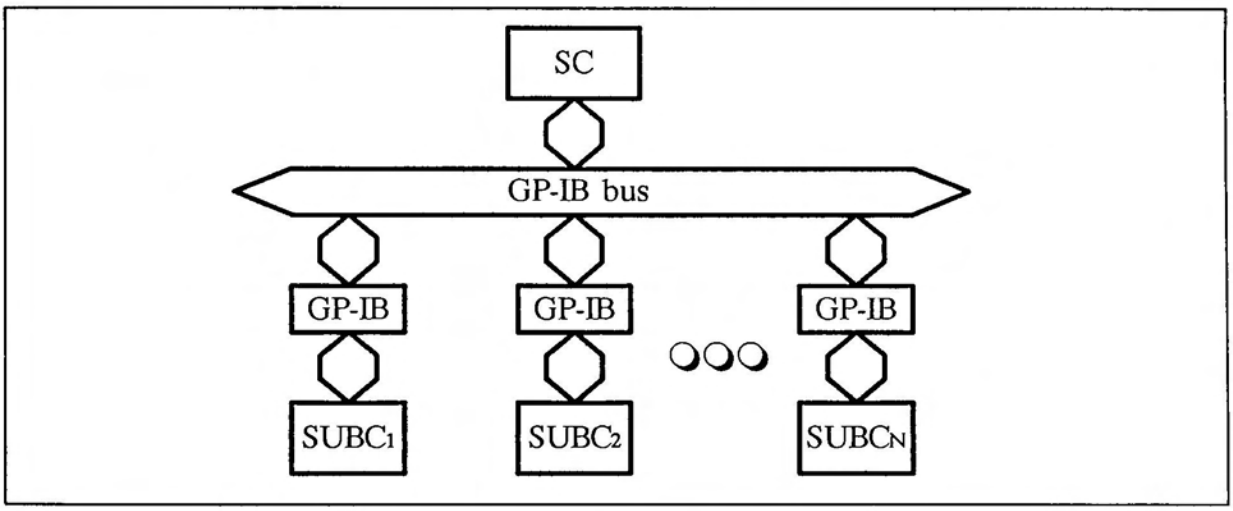

Fig.3.18. Control schema between SC and SUBC's.

3.8 Interface between HIGIPS and SC

There are three kinds of interfaces most often used in computer system. They are:

(1) RS-232C;

(2) SCSI (Small Computer System Interface);

(3) GP-IB (General Purpose Interface Bus).

RS-232C interface is the most simple and low-cost one and can be implemented most easily. But the data or messages can be only transfered over it serially.

Multiple devices can be connected to the SCSI bus and be controlled at the same protocol. A large amount of data can also be transfered over it at high-speed. But at most,

only 8 devices can be connected to this bus [120]. This restricts the efficiency and performance of the system at some extent.

In contrast with the drawbacks of these two interfaces, a large number of devices (at most 15) [120] can be attached to the GP-IB bus without any supplemental hardware. And data or message exchanges between devices can be performed at high-speed and flexibly. Hence, HIGIPS employs the GP-IB interface. The GP-IB interface scheme is shown in Fig.3.18.

Chapter 4

PROTOTYPE HIGIPS

According to the design concepts discussed in chapter 3, a prototype HIGIPS was designed and implemented [104, 105, 109, 110, 115]. Fig.4.1 shows the block diagram of prototype HIGIPS, and Fig.4.2.A-J gives the total circuit of it.

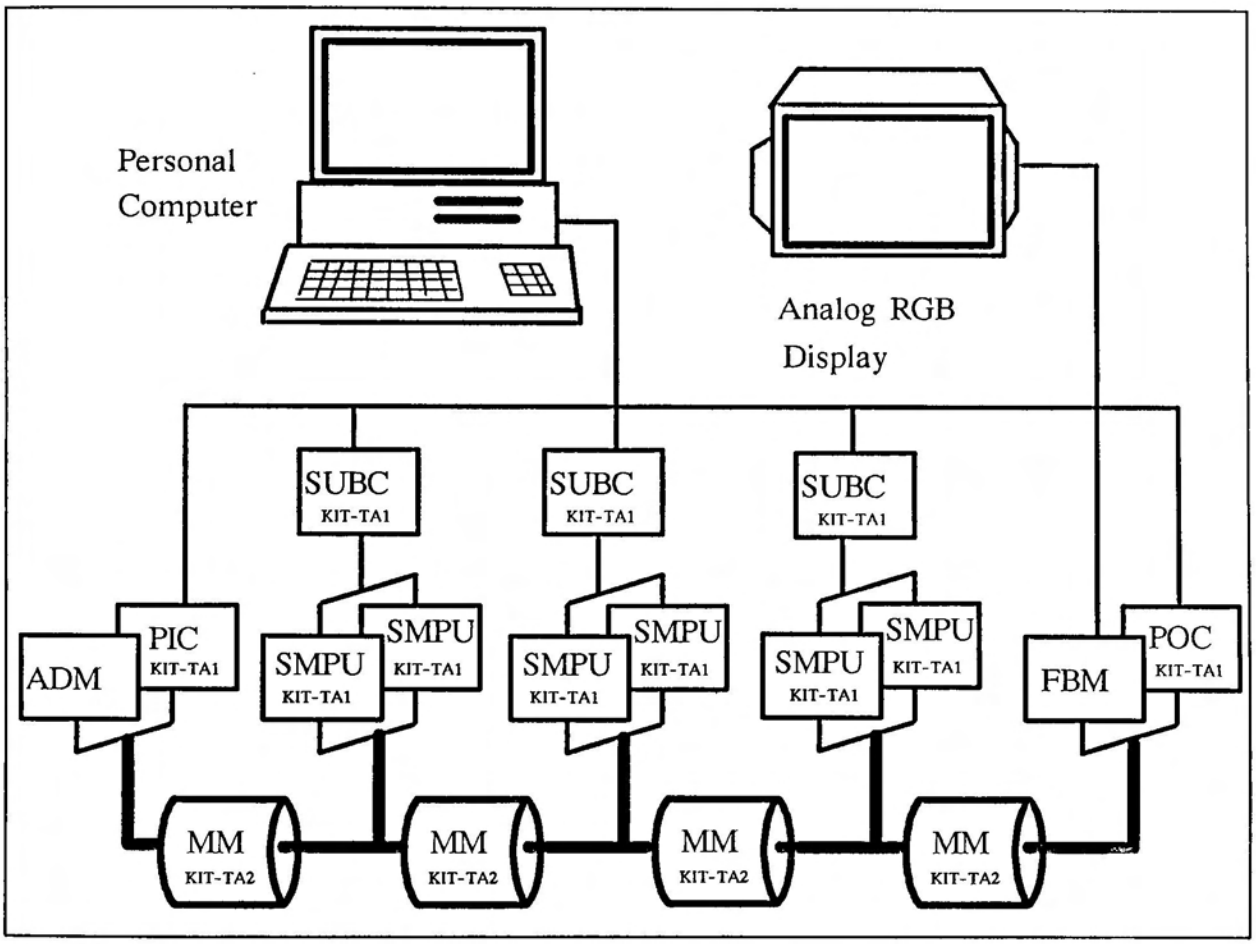

Fig.4.1. Structure of Prototype HIGIPS.

It uses three stages and each stage employs 2 PE's but it can be easily extended to 15 stages and 8 PE's in each stage. All PE's are based on the NEC μPD70216 (V50) 16-bit microprocessor. Only off-the-shelf components are used. This eliminates the need for VLSI chips and reduces development time and construction costs. By using a modular design concept, only two different boards are designed and implemented: a CPU board—KIT-TA1 (see Fig.4.2.B-D) and a memory board—KIT-TA2 (see Fig.4.2.A, E-G, J).

4.1 CPU Board — KIT-TA1

The KIT-TA1 board contains the μPD70216 microprocessor, 512 kilobytes dynamic memories as LM, parallel I/O interfaces (GP-IB and 8255), serial I/O interface (RS232C), bus controller, bus arbiter, and latches and buffers.

4.1.1 System Clock and Reset Circuit

V50 has an on-chip clock generator. There are two ways to use this resident clock generator: 1) to connect a crystal and corresponding condensers to X1 and X2 pins of V50 chip to make the inside clock generator oscillate; 2) to provide the clock to X1 pin and inverse output of that clock to X2 pin. HIGIPS employs the second method. As shown in Fig.4.2.A, 16-MHz CLK is generated by a OSC chip, and the inverse of CLK is produced by a PAL device PAL16L8A (V50_LM2) as given in Fig.4.2.B. V50 acts at 1/2CLK, i.e., 8-MHz. In addition, since each stage of HIGIPS is a multiprocessor and each V50 works on the maximum mode, another clock for the bus is needed. This bus clock BCLK is also generated by a OSC chip. The bus speed is 10-MHz.

The reset circuit consists of two parts: 1) a power-on reset; 2) a switch reset. The switch reset uses the principle of RS-flip-flop to repress the ripple formed at the moment when pressing down the reset switch.

4.1.2 Dynamic RAM Circuit

The timing control of DRAM is done by the address multiplexer composed by a PAL device PAL16L8A (V50_LM2) and two 74LS257 chips (see Fig.4.2.B). As the V50 chip also includes a refresh control unit, it is easy to design the timing using the RAS-only-refresh mode if to utilize DRAM chip with the capacity less than or equal to 256 kilobits. The gate delay time of PAL16L8A is used to generate DRAM chip's timings.

Table 4.1 shows the CUPL2 [117] logic equations for the PAL device. $\overline{RAS}$ is made from $\overline{MWR}$ and $\overline{MRD}$. $\overline{CASH}$, $\overline{CASL}$ and SLX are made from the delayed $\overline{RAS}$ and address lines A17-A19. $\overline{CASH}$ and $\overline{CASL}$ are the $\overline{CAS}$ of the odd address and even address, correspondingly.

The address signals are multiplexed by two 74LS257 chips. They multiplex only 8-bit addresses, there needs one more address line (A8) for accessing 256 kilobits DRAM chips. A8 is also made by this PAL device. $\overline{MWR}$ is directly used as $\overline{WR}$ of DRAM chips.

$\overline{RAS}$, $\overline{CASH}$ and $\overline{CASL}$ are serialized with a 22Ω dumping resister, respectively. This is to reduce the Q of equivalent oscillating circuit formed by the wiring inductance (L) of

the print circuit and the gate capacitance (C) of DRAM chips, and to restrain the ripple. Each signal line is pulled up by a 2.2KΩ at the end of it. This is also to repress the ripple formed when the signal alternates (from H to L or vice versa).

ZIP-type DRAM chips (MB81256-12ZIP) are utilized to shorten the wiring length and decrease the wiring inductance. DRAM access timings are shown in Fig.4.3. The timing parameters are given in table 4.2.

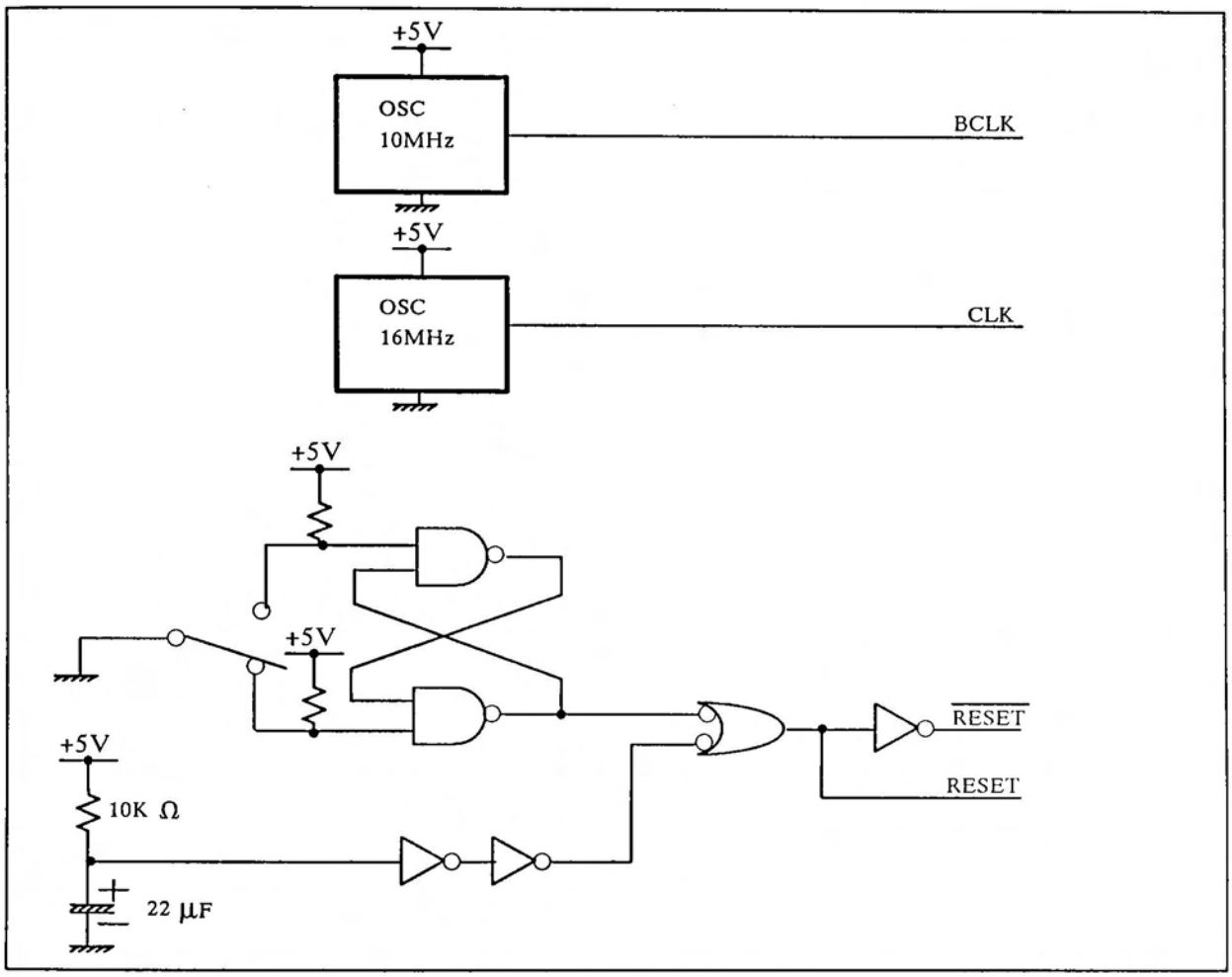

Fig.4.2.A. HIGIPS circuit: the oscillator and reset circuit.

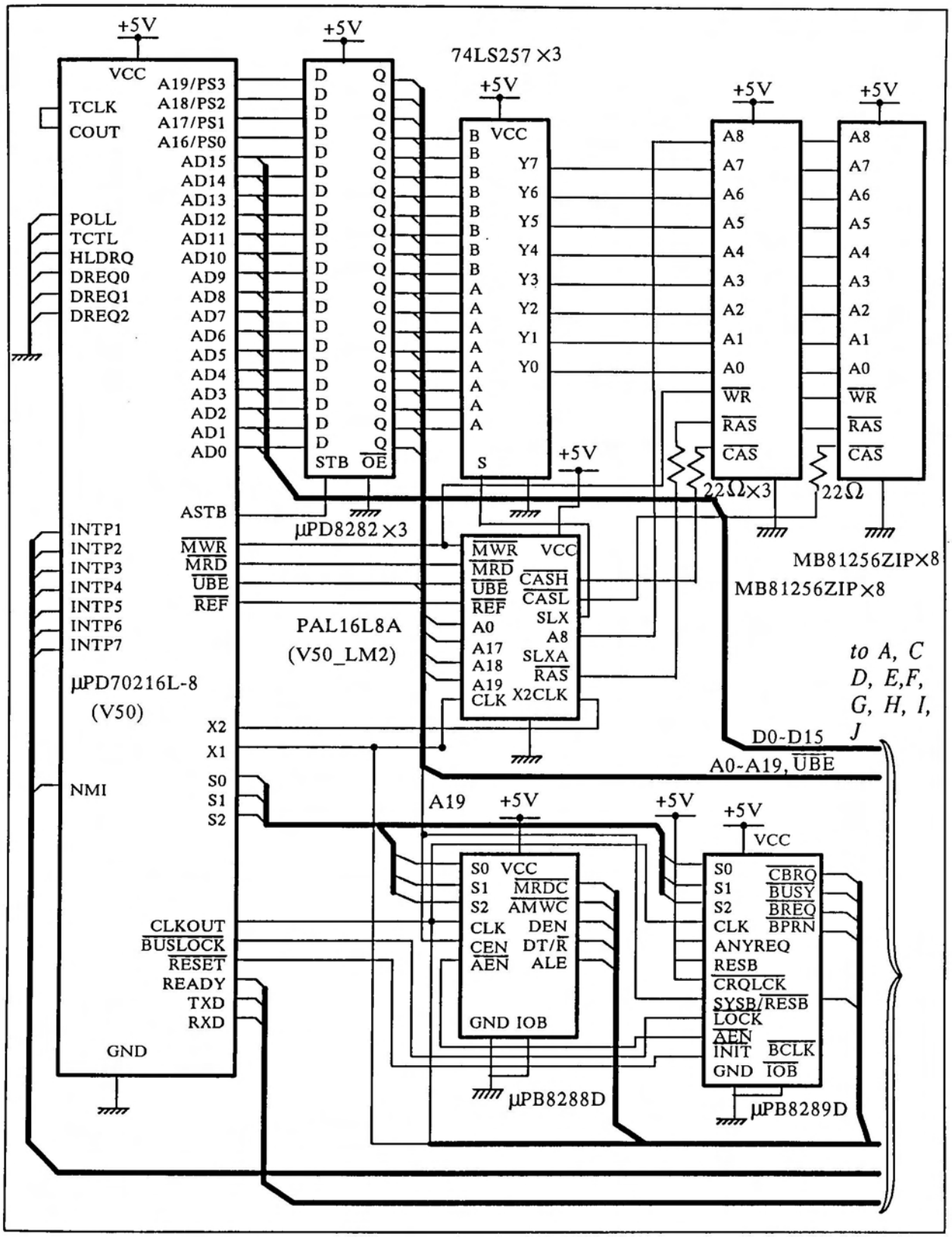

Fig.4.2.B. HIGIPS circuit: CPU and local memory.

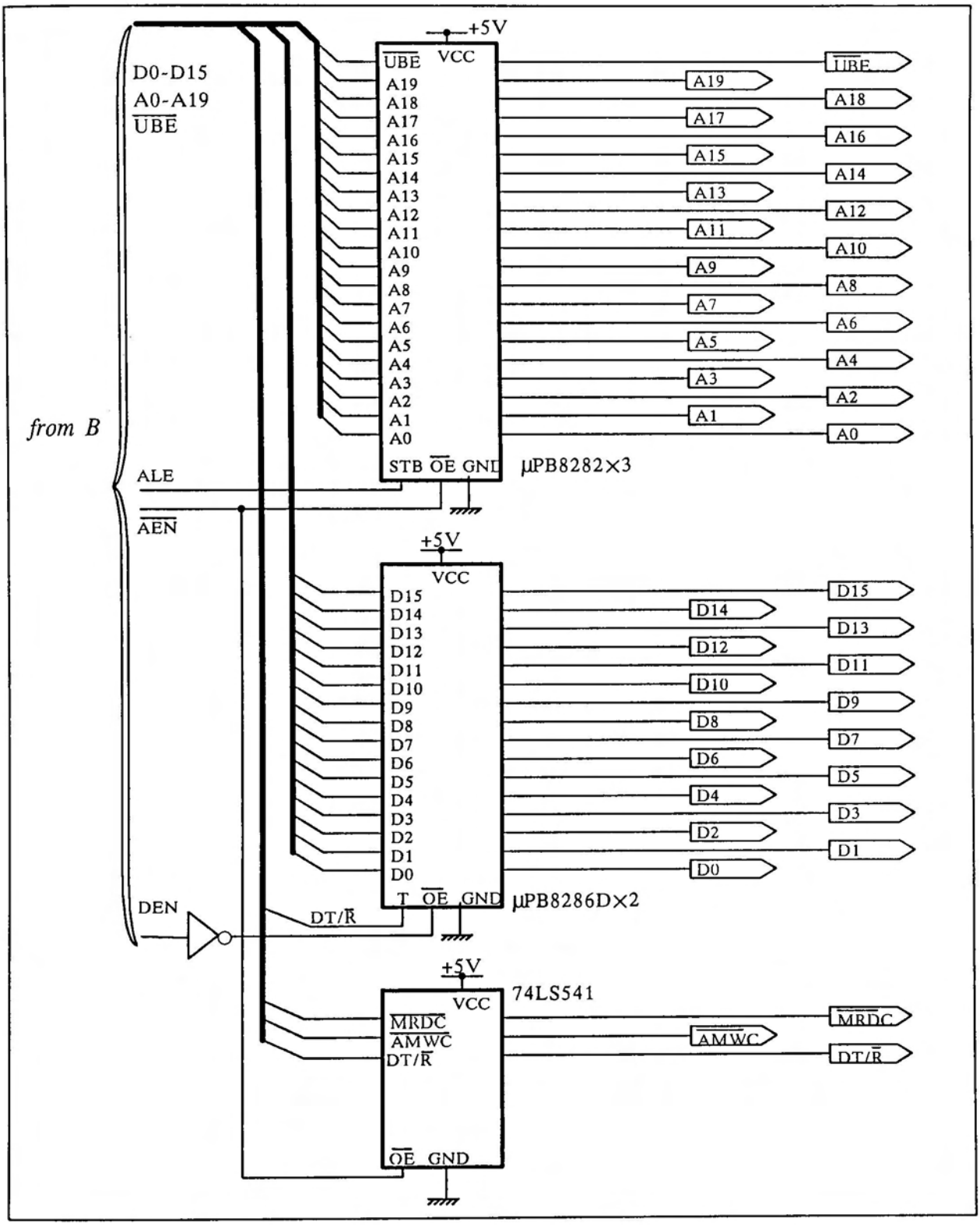

Fig.4.2.C. HIGIPS circuit: bus latches and buffers.

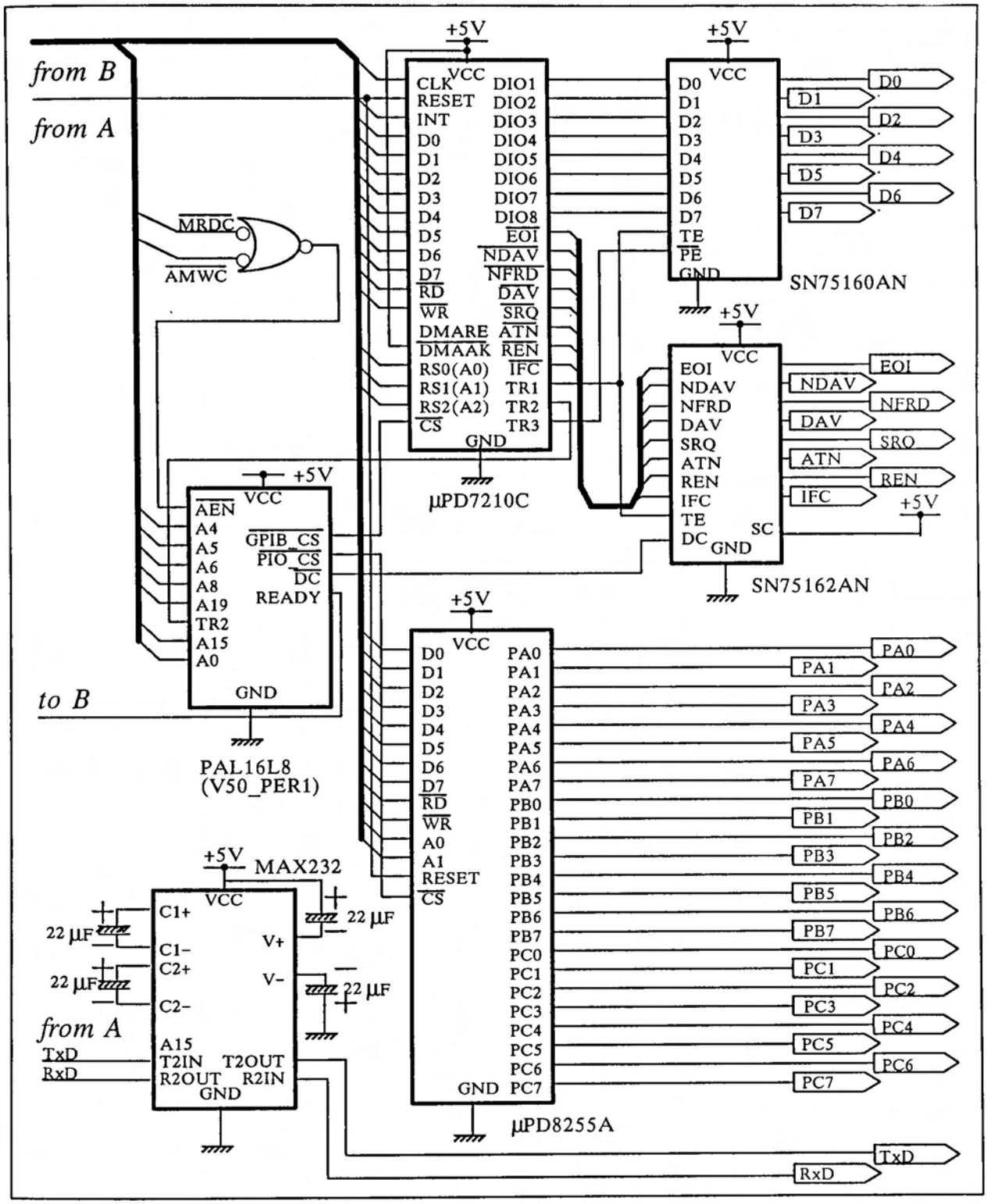

Fig.4.2.D. HIGIPS circuit: the parallel interface.

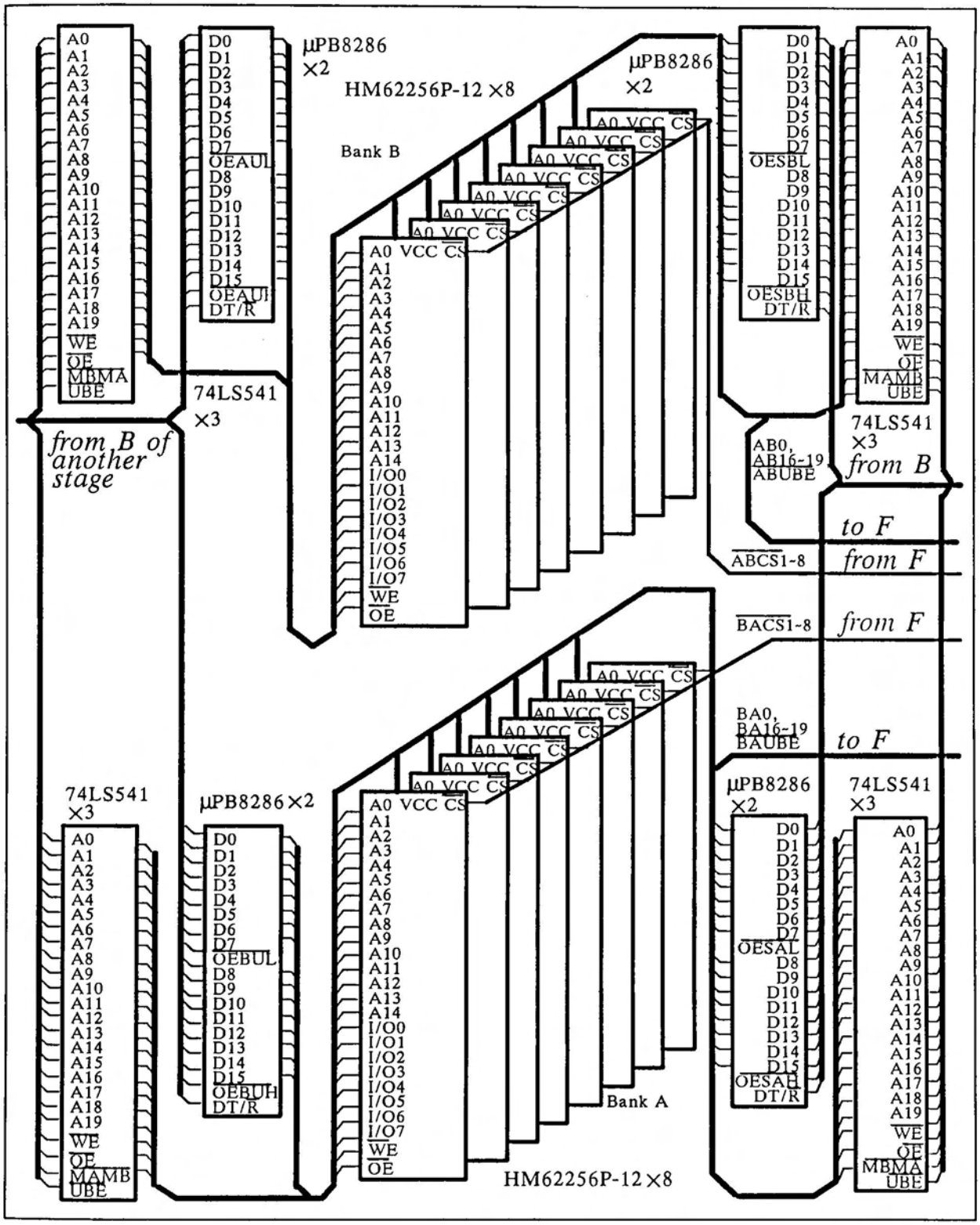

Fig.4.2.E. HIGIPS circuit: the common memory.

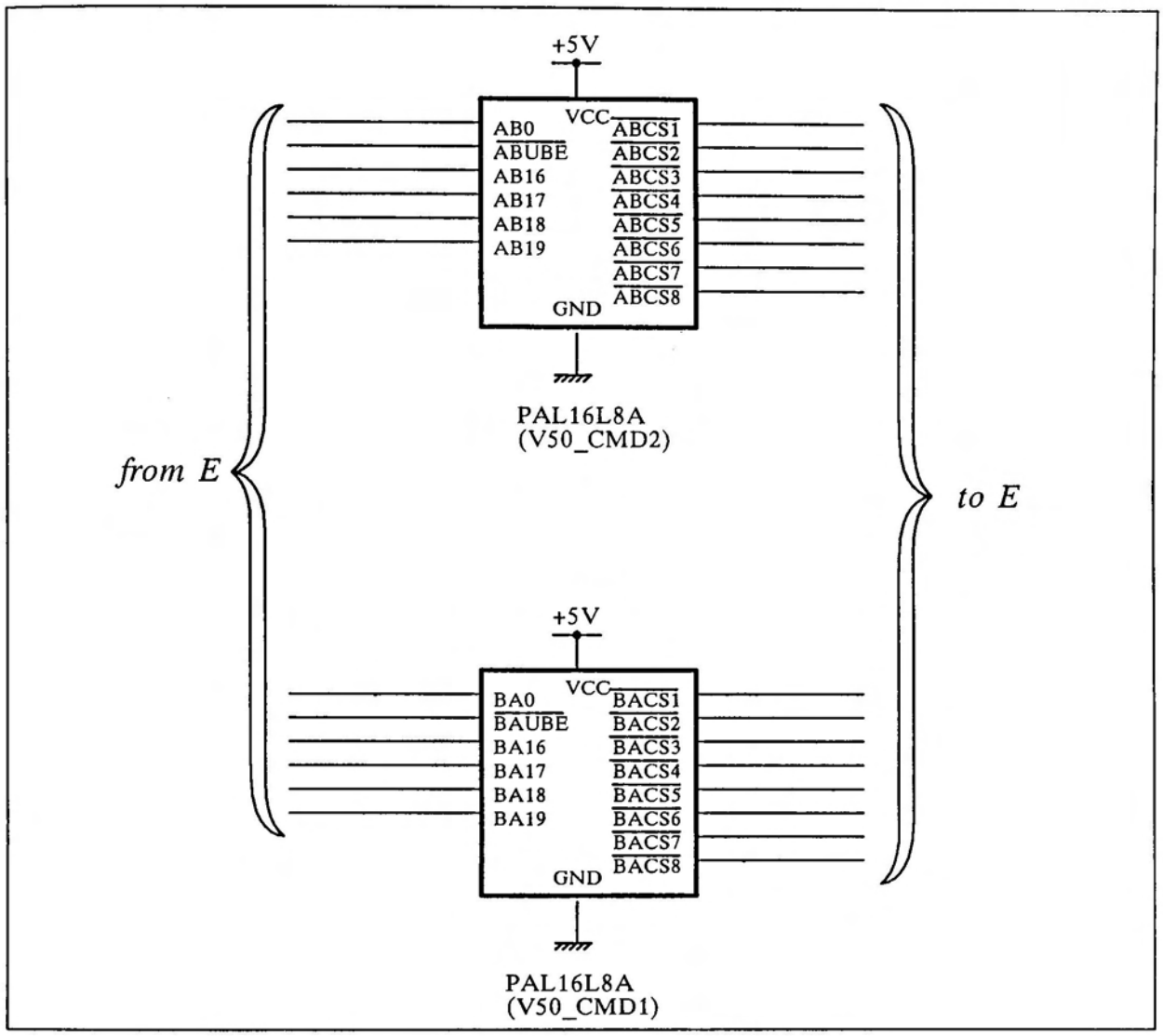

Fig.4.2.F. HIGIPS circuit: the common memory chip selects.

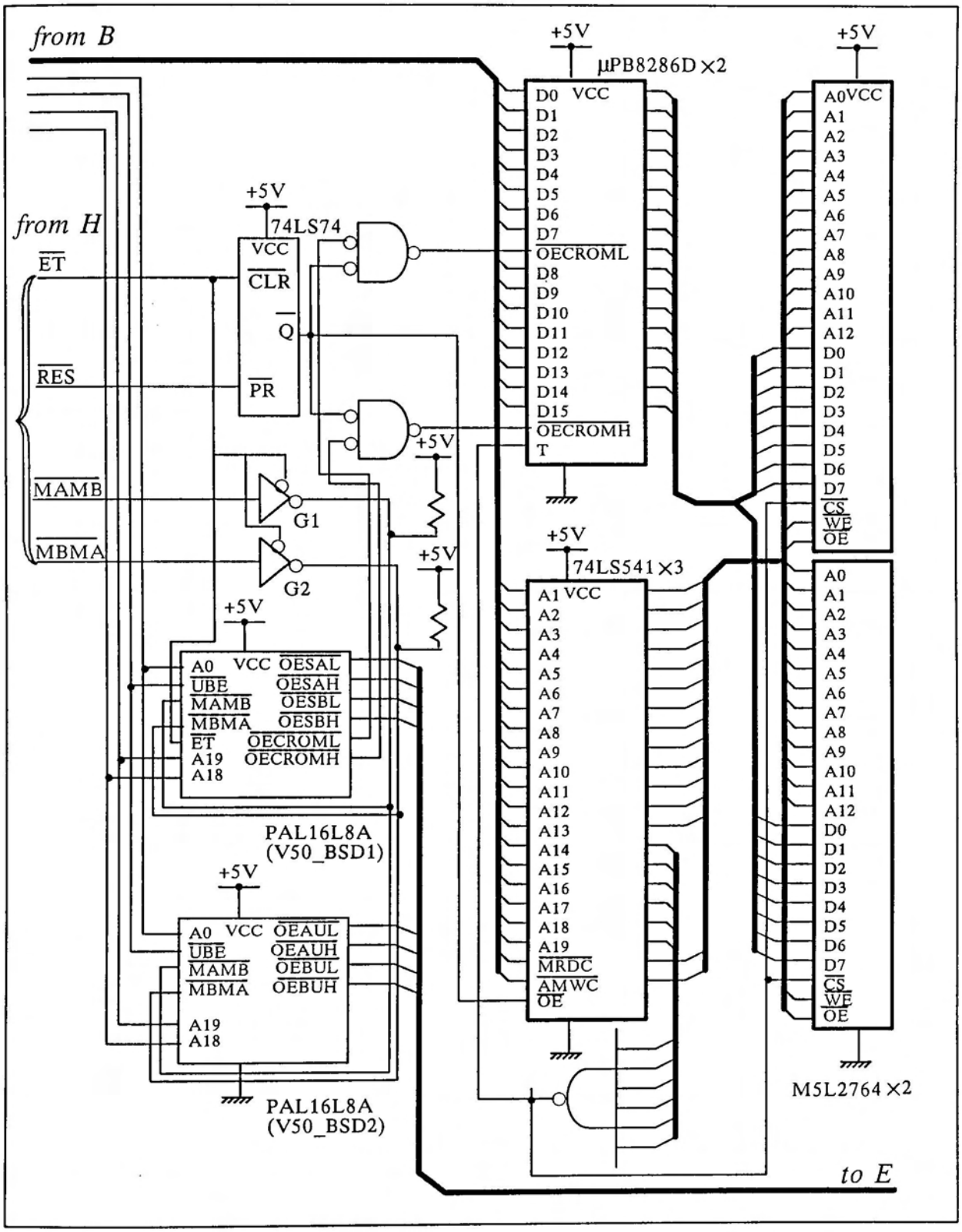

Fig.4.2.G. HIGIPS circuit: the common ROM.

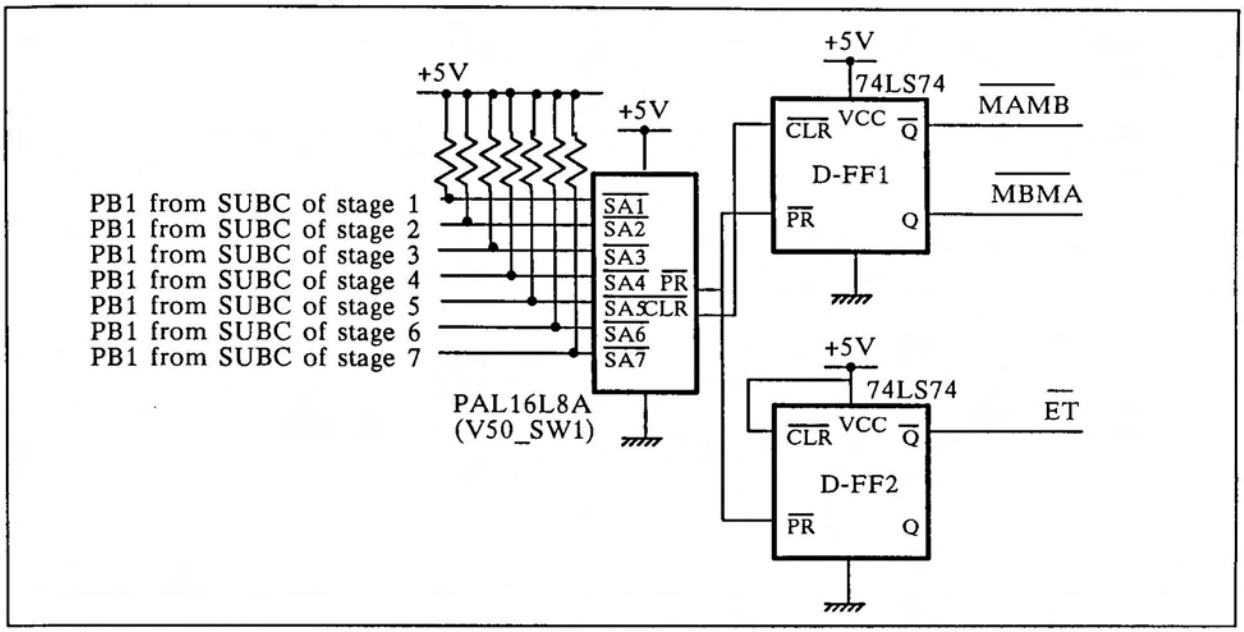

Fig.4.2.H. HIGIPS circuit: automatic on/off control of bus switches.

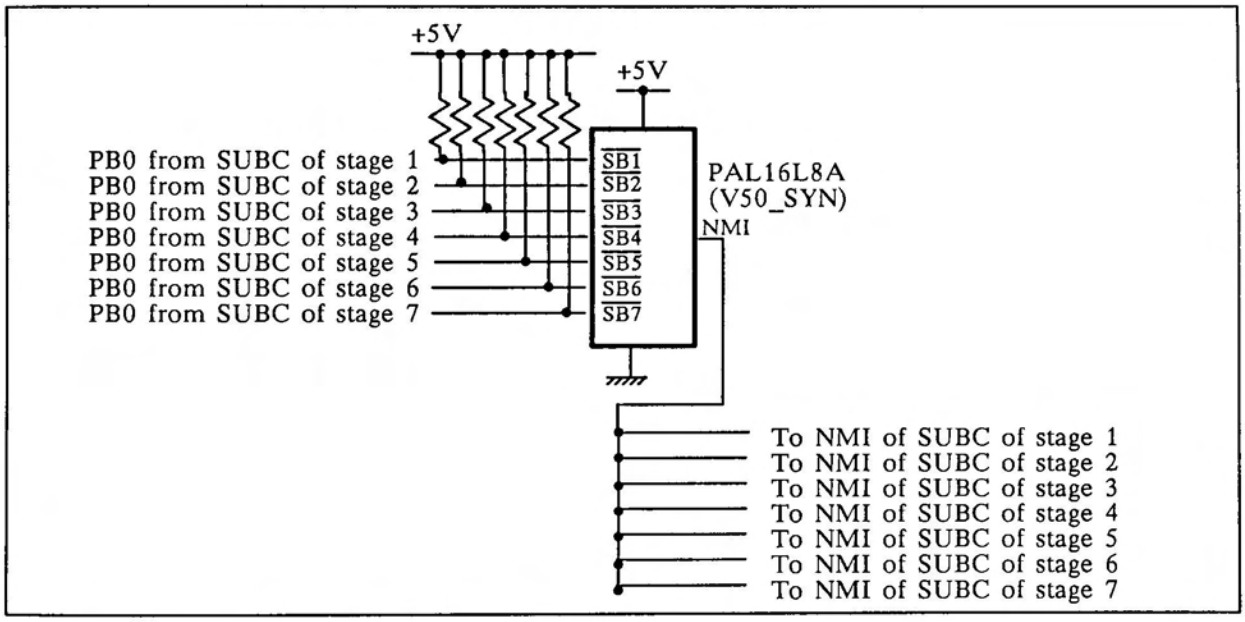

Fig.4.2.I. HIGIPS circuit: restarting of the whole system.

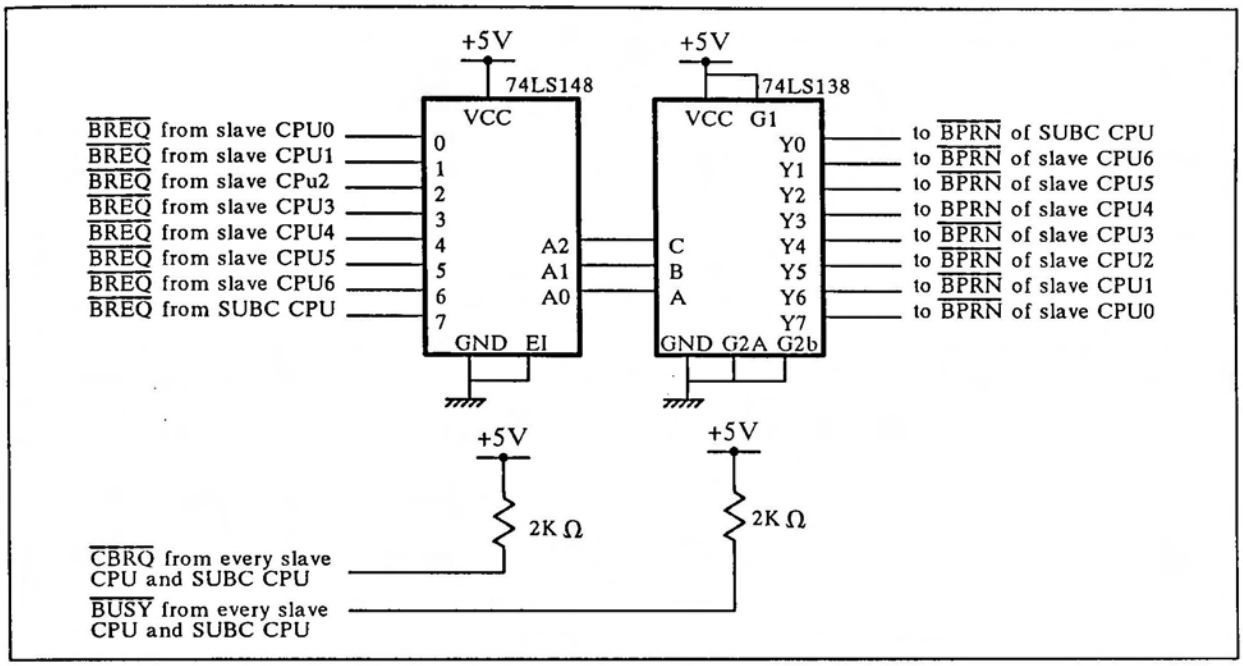

Fig.4.2.J. HIGIPS circuit: the bus arbitration and bus control.

Table 4.1

Logic Equations of PAL Device for DRAM Control Timing

```
Name            V50_LM2;
Partno          I don't know;
Date            07/25/88;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        V50 Board;
Location        LOC;
/*************************************************************************/
/* This device generates the !RAS, !CASH, !CASL, SLX etc. signals        */
/* for local memory.                                                     */
/*************************************************************************/
/* Allowable target device types: PAL16L8                                */
/*************************************************************/
/** Inputs **/
Pin [1,2]       = ![mwr, mrd]       ; /* Memory data strobes             */
Pin 3           = !ube              ; /* Upper byte enable               */
Pin 4           = !ref              ; /* Dynamic RAM refresh signal      */
Pin 5           = a0                ; /* CPU address bus                 */
Pin [6..8]      = [a17..a19]        ; /* CPU address bus                 */
Pin 9           = clk               ; /* Clock from OSC                  */
/** Outputs **/
Pin 13 = !ras               ; /* RAS for DRAM                    */
Pin 18 = !cash              ; /* CAS for DRAM of odd address     */
Pin 17 = !casl              ; /* CAS for DRAM of even address    */
Pin 16 = slx                ; /* Multiplexer strobe              */
Pin 15 = a8                 ; /* Address bus                     */
Pin 14 = slxa               ;
Pin 12 = x2clk              ; /* Clock to X2 pin of V50          */
/** Declarations and intermediate variable definitions **/
/** Logic equations **/
!ras    = !(!(!mwr) # !(!mrd))      ; /* RAS for row address     */
slxa    = !(!ras)                   ;
slx     = slxa                      ;
!cash   = !(!(!ube) & !ref & !a19 & slx)        ;
!casl   = !(!a0 & !ref & !a19 & slx)            ;
a8      = !((!a17 & slx) # (!a18 & !slx))       ;
x2clk   = !clk                                  ;
```

Note: !A means the inverse of A. And this is true through the whole of this paper.

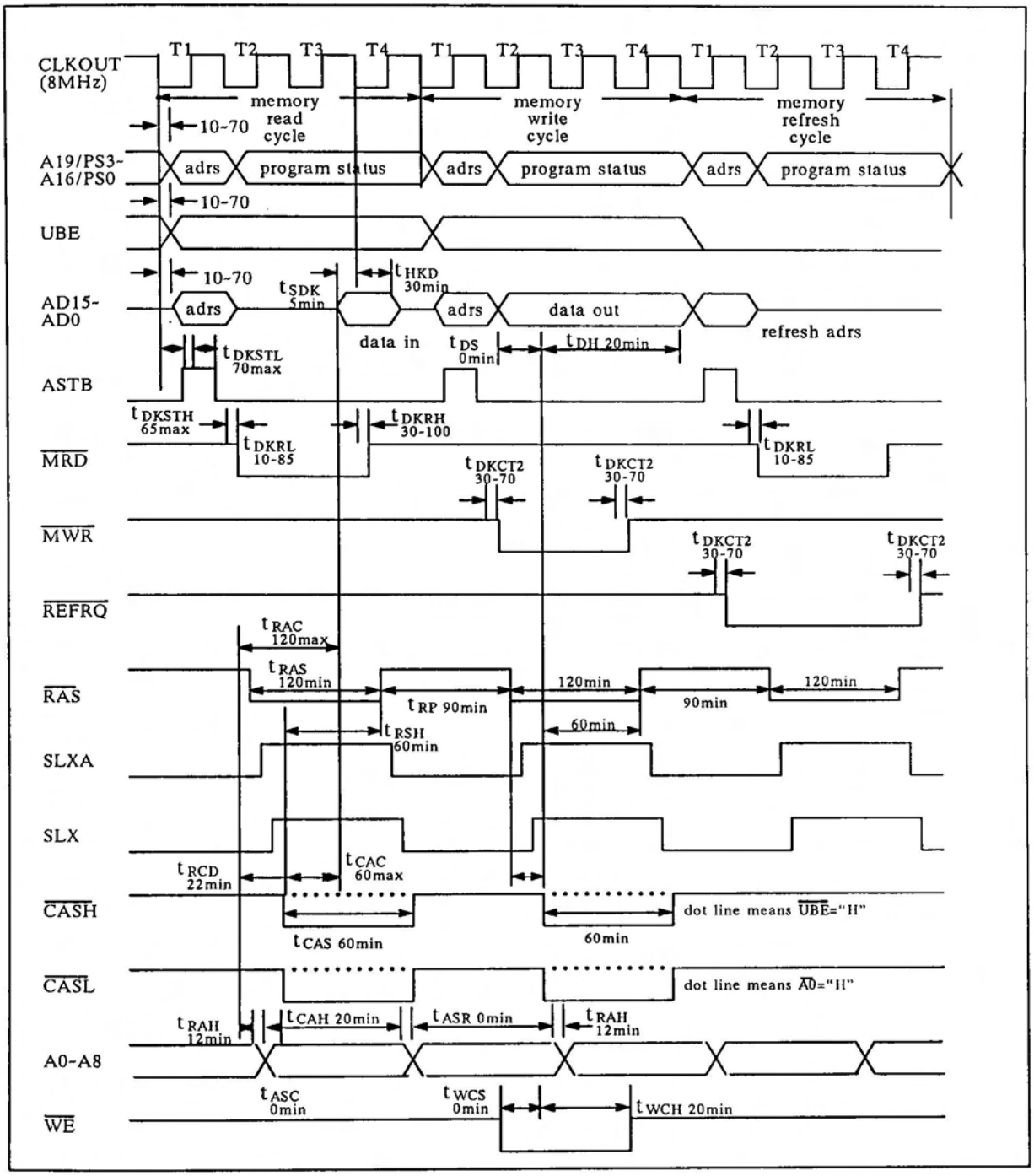

Fig.4.3. Timing charts of DRAM circuit.

Table 4.2
Dynamic RAM (MB81256-12ZIP) Timing Parameters

items	symbols	standards(ns)
row address set-up time	t_{ASR}	0min
row address hold time	t_{RAH}	12min
column adress set-uptime	t_{ASC}	0min
column adress hold time	t_{CAH}	20min
!RAS, !CAS delay time	t_{RCD}	22min
!RAS pulse width	t_{RAS}	120min
!RAS precharge time	t_{RP}	90min
!CAS pulse width	t_{CAS}	60min
!RAS hold time	t_{RSH}	60min
!RAS access time	t_{RAC}	120max
!CAS access time	t_{CAC}	60max
write cycle set-up time	t_{WCS}	0min
write cycle hold time	t_{WCH}	20min
data read set-up time	t_{DS}	0min
data read hold time	t_{DH}	20min

4.1.3 Bus Arbitration and Bus Control

The V50 CPU works at maximum mode, so the bus control signal and bus arbitration are necessary. The bus arbiter is μPB8289D chip which applies and obtains the priority of using the common bus. The bus controller is μPB8288D that provides all the control signals for accessing the common memory.

Fig.4.2.J gives the parallel priority circuit. $\overline{\text{BREQ}}$'s from all the slave CPU's and SUBC of the same stage are fed to the encoder (74LS148) which grades the inputs into 8 levels represented by 3-bit addresses. These 3-bit addresses are sent to the decoder (74LS138). The outputs of the decoder (Y0-Y7) are fed to $\overline{\text{BPRN}}$ of each slave CPU and SUBC CPU, respectively. Only one output of decoder can be active "0", and the corresponding CPU connected with this output can access the common memory. Note here that the Y0 is fed to $\overline{\text{BPRN}}$ of SUBC CPU to give it the highest priority, i.e., SUBC CPU can have the priority to use the common bus anytime.

4.1.4 Local Bus Latches and Buffers

Each CPU can use the common bus via the latches, transceivers and buffers (see Fig.4.2.B). The address bus A0-A19 and $\overline{UBE}$ are latched by the latch chips (μPB8282), the data bus is driven by the transceiver chips (μPB8286), and the memory control signals $\overline{MRDC}$, $\overline{AMWC}$ and DT/$\overline{R}$ are fed to the common bus via a buffer (74LS541). The control signals of these latch, transceiver and buffer chips are provided by the bus controller, concretely, ALE to STB; $\overline{AEN}$ to $\overline{OE}$'s of the latch chips and buffer chip; DT/$\overline{R}$ to T and the inverse of DEN to $\overline{OE}$'s of the transceiver chips.

4.1.5 I/O Devices

Each PE of V50 has three kinds of interfaces: 1) RS-232C; 2) GP-IB; 3) programmable peripheral interface 8255 (see Fig.4.2.D). As V50 CPU has the resident serial control unit (SCU) which is a subset of ordinary programmable communication interface 8251, to construct RS-232C serial interface needs nothing except to employ a transceiver chip (MAX232) externally for changing the signal level.

RS-232C interface is used only for hardware and software debugging. After this debugging, all the controls are performed via the GP-IB interface which is constructed by the GP-IB controller chip (μPD7210C) and bus transceiver chips (SN75160AN and SN75162AN). Only SUBC in a stage has GP-IB interface.

Parallel interface 8255 (μPD8255A) is for providing the signals for automatic restarting of processing program and automatic *on/off* of bus switches. This will be explained in section 4.2.

The chip select signals of GP-IB controller μPD7210C and programmable parallel interface μPD8255A and READY of V50 CPU are formed by the PAL16L8A (V50_PER1) device. Table 4.3 gives their logic equations, Table 4.4 displays the I/O accessing timing parameters, and Fig.4.4 shows the I/O access timing.

Table 4.3

Logic Equations of PAL Device for I/O Chip Selects and CPU-Ready

```
Name            V50_PER1;
Partno          I don't know;
Date            07/26/88;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        ASSY;
Location        LOC;
/*************************************************************************/
/* This device generates the control signals for peripheal LSI's such as  */
/* !GPIB_CS, !PIO_CS, READY and so on.                                     */
/*************************************************************************/
/* Allowable target device types: PAL16L8                                  */
/*************************************************************************/
/** Inputs **/
Pin 1           = !aen              ; /* Address enable                   */
Pin [2..4,5,6]  = [a4..a6,a7,a19]; /* CPU address bus                     */
Pin 7           = tr2               ; /* GP-IB bus transceiver strobe     */
Pin 8           = a15               ; /* CPU address bus                  */
Pin 9           = a0                ; /* CPU address bus                  */
/** Outputs **/
Pin [19,18]     = ![gpib_cs,pio_cs]; /* Chip selects                      */
Pin 17 = !dc                ; /* GP-IB bus transceiver enable             */
Pin 16 = ready              ; /* CPU ready                                */
/** Declarations and intermediate variable definitions **/
/** Logic equations **/
!gpib_cs        = !(!a15 & a8 & !a6 & !a5 & a4 & !a0);
!pio_cs = !(!a15 & a8 & !a6 & !a5 & !a4 & !a0);
!dc     = !tr2;
ready   = !a19 # (a19 & !(!aen));
```

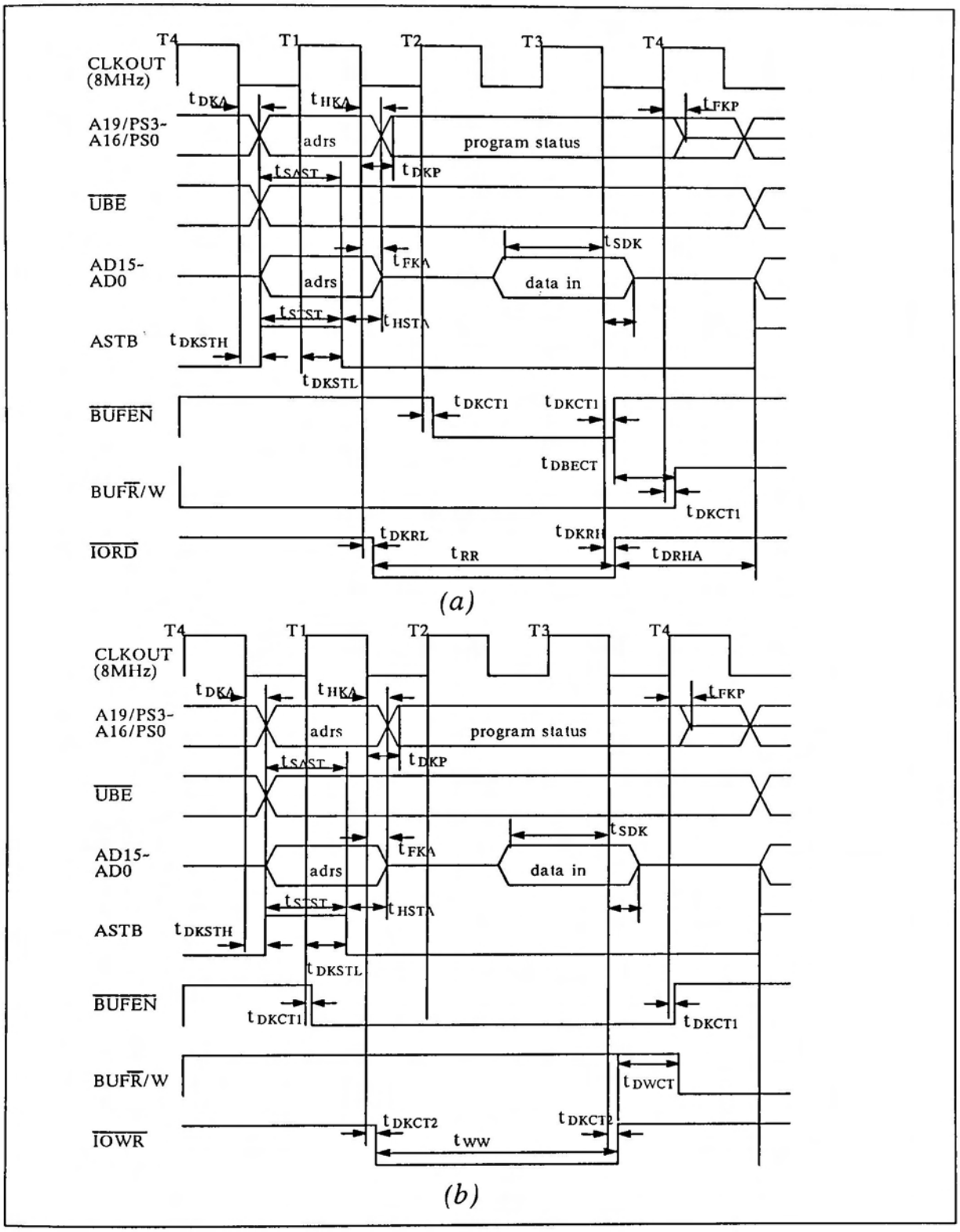

Fig.4.4. (a) I/O read timing chart; (b) I/O write timing chart.

Table 4.4
Characteristics of I/O Accessing Timing Chart

Items	Symbols	Condition	Standards (nS)	
clock cycle	t_{CYK}		124min	500min
"H" width of clock	t_{KKH}		0.5 t_{CYK} -7	
"L" width of clock	t_{KKL}		0.5 t_{CYK} -7	
set data time	t_{SDK}		5min	
data hold time	t_{HKD}		30min	
address delay time	t_{DKA}		10min	70max
address hold time	t_{HKA}		10min	
program status delay time	t_{DKP}		10min	60max
program float delay time	t_{FKP}		10min	60max
address set-up time	t_{SAST}		t_{KKL} -30	
address float delay time	t_{FKA}		1HKA	60max
ASTB delay time 1	t_{DKSTH}			65max
ASTB delay time	t_{DKSTL}			70max
ASTB width	t_{STST}		t_{KKL} -10	
address hold time (ASTB)	t_{HSTA}		t_{KKH} -30	
control 1 delay time	t_{DKCT1}		30min	90max
control 2 delay time	t_{DKCT2}		30min	90max
address float—!RD delay time	t_{DAFRL}		0min	
!RD delay time	t_{DKRL}		10min	85max
!RD delay time	t_{DKRH}		25min	100max
address delay time	t_{DRHA}		t_{CYK} -65	
!RD "L" width	t_{RR}		2t_{CYK} -50	
!BUFNE—BUFR/W delay time	t_{DBECT}	read cycle	t_{KKL} -20	
data output delay time	t_{DKD}		10min	60max
data float delay time	t_{FKD}		10min	60max
!WR "L" width	t_{WW}		2t $_{CYK}$ -40	
!WR—!BUFNE delay time	t_{DWCT}	write cycle	t_{KKL} -20	

Notes: Control 1 means !BUFEN, BUF!R/W;
Control 2 means !MWR, !IOWR, !INTAK, !REFRA.

4.2 Differences between SUBC and SMPU

SUBC and SMPU can be implemented by the same KIT-TA1 board. The differences are: 1) SMPU does not use GP-IB interface; 2) there exist some different usages of interrupt lines and the outputs of programmable parallel interface 8255. Table 4.5 illustrates the details, where PA0 means the first line of port A and PC_k means the *k*-th line of port C of 8255 interface.

Table 4.5
Difference between SUBC and SMPU's

PERIPHERAL DEVICES	PIN'S	SUBC	SMPU'S
INTERRUPT CONTROL UNIT	INTP1	connected with PA0 of SMPU 1	connect with PC_k
	INTP2	connected with PA0 of SMPU 2	ground
	INTP3	connected with PA0 of SMPU 3	ground
	INTP4	connected with PA0 of SMPU 4	ground
	INTP5	connected with PA0 of SMPU 5	ground
	INTP6	connected with PA0 of SMPU 6	ground
	INTP7	connected with PA0 of SMPU 7	ground
	NMI	connected with the output of automatic restarting circuit	ground
PROGRAMMABL PARALLEL INTERFACE 8255	PA0-PA7	for reading the status of DIP switch on board	for reading the status of DIP switch on board
	PB0	to automatic restarting circuit	no use
	PB1	to automatic restarting circuit	no use
	PB2-PB6	no use	no use
	PB7	board status indicator	board status indicator
	PC0	to NMI of SMPU 1 inside stage	no use
	PC1	to NMI of SMPU 2 inside stage	no use
	PC2	to NMI of SMPU 3 inside stage	no use
	PC3	to NMI of SMPU 4 inside stage	no use
	PC4	to NMI of SMPU 5 inside stage	no use
	PC5	to NMI of SMPU 6 inside stage	no use
	PC6	to NMI of SMPU 7 inside stage	no use
	PC7	no use	no use
GP-IB INTERFACE		in use	no use

Note: k is corresponding to the CPU number get from DIP switch on board.

4.3 Pin Assignments of KIT-TA1 Board

The size of KIT-TA1 board is 226mm×288mm. There is a 100-pin edge connector on the right side and a 26-pin GP-IB connector on the left side when located the KIT-TA1 board as in Fig.4.5.

The top view side of Fig.4.5 is for settling IC chips and the inverse side is for soldering. The pins on the top view side and inverse side are numbered from BR1 to BR50, and AR1 to AR50, respectively. The meanings of these pins are explained in table 4.6.

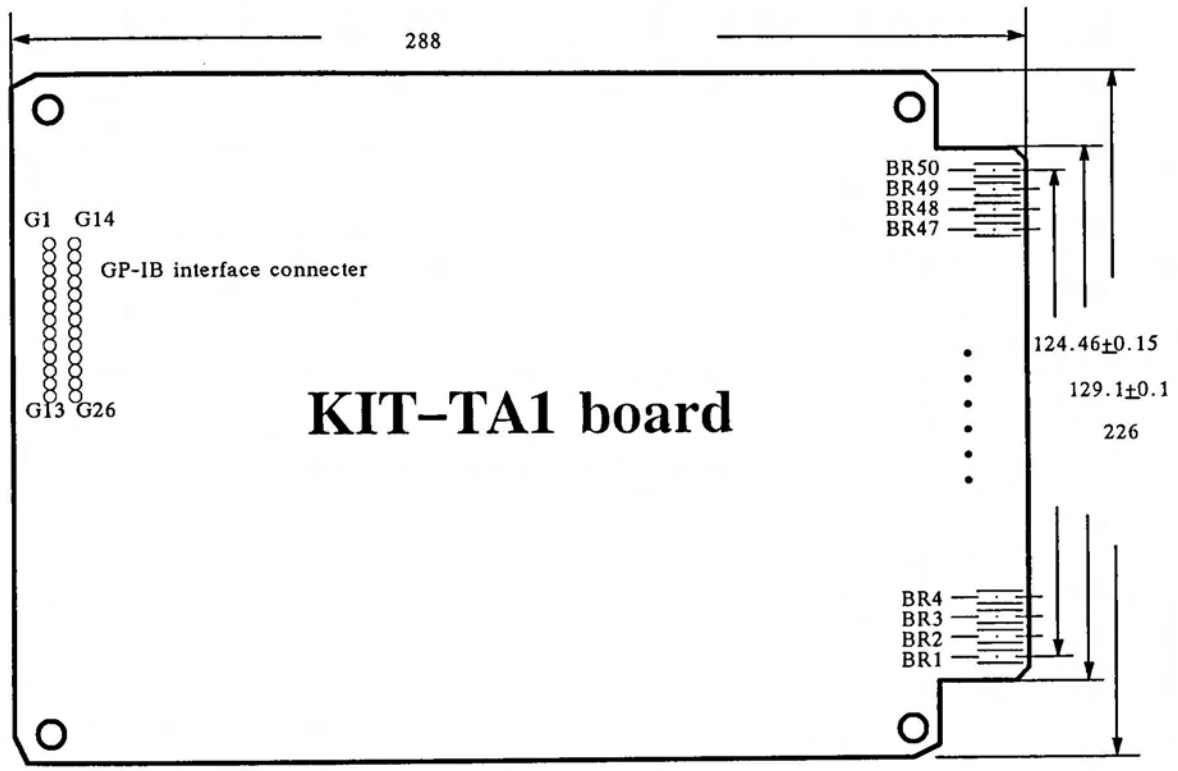

Fig.4.5. Outward appearance of KIT-TA1 board.

Table 4.6

Pin Assignments and Their Functions of KIT-TA1 Board

No.	SYBLS.	DIR.	FUNCTIONS
AR1	GND		
AR2	GND		
AR3	GND		
AR4	A9	3SO	Address bus
AR5	A10	3SO	Address bus
AR6	A11	3SO	Address bus
AR7	A12	3SO	Address bus
AR8	A13	3SO	Address bus
AR9	A14	3SO	Address bus
AR10	A15	3SO	Address bus
AR11	A16	3SO	Address bus
AR12	A17	3SO	Address bus
AR13	A19	3SO	Address bus
AR14			
AR15			
AR16			
AR17			
AR18			
AR19			
AR20			
AR21			
AR22			
AR23			
AR24			
AR25			
AR26			
AR27	D0	I/O	Data bus
AR28	D1	I/O	Data bus
AR29	D2	I/O	Data bus
AR30	D3	I/O	Data bus
AR31	D4	I/O	Data bus
AR32	D5	I/O	Data bus
AR33	D6	I/O	Data bus
AR34	D7	I/O	Data bus
AR35	A0	3SO	Address bus
AR36			
AR37			
AR38			
AR39			
AR40	RESET	I	Active high reset
AR41			
AR42			
AR43			
AR44			
AR45			
AR46			
AR47			
AR48	VCC		
AR49	VCC		
AR50	VCC		

No.	SYBLS.	DIR.	FUNCTIONS
BR1	GND		
BR2	GND		
BR3	GND		
BR4	A1	3SO	Address bus
BR5	A2	3SO	Address bus
BR6	A3	3SO	Address bus
BR7	A4	3SO	Address bus
BR8	A5	3SO	Address bus
BR9	A6	3SO	Address bus
BR10	A7	3SO	Address bus
BR11	A8	3SO	Address bus
BR12	A18	3SO	Address bus
BR13			
BR14			
BR15			
BR16			
BR17			
BR18			
BR19			
BR20			
BR21			
BR22			
BR23			
BR24			
BR25			
BR26			
BR27	D8	I/O	Data bus
BR28	D9	I/O	Data bus
BR29	D10	I/O	Data bus
BR30	D11	I/O	Data bus
BR31	D12	I/O	Data bus
BR32	D13	I/O	Data bus
BR33	D14	I/O	Data bus
BR34	D15	I/O	Data bus
BR35	!UBE	3SO	Upper byte enable
BR36			
BR37			
BR38	CLK	I	CPU clock
BR39	BCLK	I	Bus clock
BR40	!RESET	I	Active low reset
BR41	!WR	3SO	Memory write enable
BR42	!RD	3SO	Memory read enable
BR43	DT!/R	3SO	Buffer write/read
BR44			
BR45	!CBREQ	I/O	Wired 'or' of all !CBREQ
BR46	!BUSY	I/O	Wired 'or' of all !busy
BR47			
BR48	VCC		
BR49	VCC		
BR50	VCC		

to be continued

continue

G2	DIO1	I/O	Data bus	G15	DIO5	I/O	Data bus
G3	DIO2	I/O	Data bus	G16	DIO6	I/O	Data bus
G4	DIO3	I/O	Data bus	G17	DIO7	I/O	Data bus
G5	DIO4	I/O	Data bus	G18	DIO8	I/O	Data bus
G6	EOI	3SO	End or Identify	G19	REN	3SO	Remote Enable
G7	DAV	3SO	Data Valid	G20,G1	GND	3SO	
G8	NRFD	3SO	Not Ready for Data	G21,G1	GND	3SO	
G9	NDAC	3SO	Not Data Accepted	G22	GND	3SO	
G10	IFC	3SO	Interface	G23	GND	3SO	
G11	SRQ	3SO	Service Request	G24	GND	3SO	
G12	ATN	3SO	Attention	G25	GND	3SO	
G13	SHIELD			G26	GND		

4.4 Memory Module Board — KIT-TA2

The KIT-TA2 board includes two 256-kilobyte static RAM block with bus switches to play the role of dual port memories, and 16-kilobyte EPROM which is used for the system startup, and is disconnected when the system becomes steady.

4.4.1 Common Image Memory

The common memory is segmented into two memory blocks (see Fig.4.2.E). Each is with the capacity of 256 kilobytes. This value is the maximum image data that HIGIPS can deal with.

Each block is composed of eight 256-kilobyte static RAM chips (HM62256P-12). Moreover, a pair of address bus latches (74LS541×3) and data bus transceivers (μPB8286B ×2), named as bus switches (BS and BS's, respectively), is attached to each block. This two pairs of bus switches makes the common memory operate as a dual port memory. The dual port memory function is obtained by setting the bus switches *on/off* alternately, i.e., when $\overline{\text{OESAL}}$, $\overline{\text{OESAH}}$ and $\overline{\text{MAMB}}$ are active (L), $\overline{\text{OESBL}}$, $\overline{\text{OESBH}}$ and $\overline{\text{MBMA}}$ are passive (H), which is called phase **A** of the common memory, and when $\overline{\text{OESBL}}$, $\overline{\text{OESBH}}$ and $\overline{\text{MBMA}}$ are active (L), $\overline{\text{OESAL}}$, $\overline{\text{OESAH}}$ and $\overline{\text{MAMB}}$ are passive (H), which is called phase **B**. The common memory accessing timing charts are shown in Fig.4.6.

The memory chip select signals for these two memory blocks are generated by the PAL devices PAL16L8A (V50_CMD1) and PAL16L8A (V50_CMD2). Table 4.7 and table 4.8 present their logic equations correspondingly.

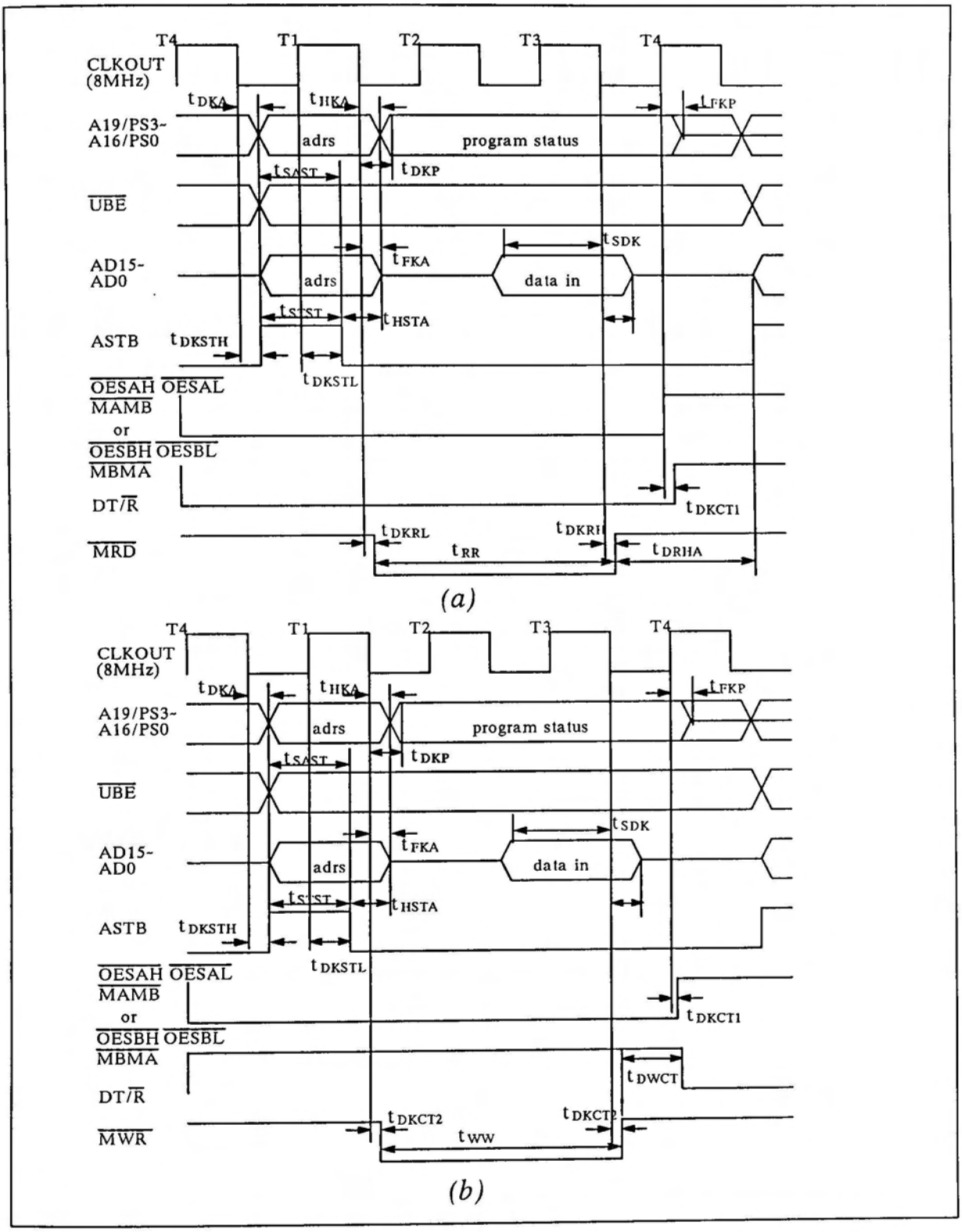

Fig.4.6. (a) Memory read timing chart; (b) Memory write timing.

Table 4.7

Logic Equations of PAL Device for Common Memory (Bank A) Chip Selects

```
Name            V50_CMD1;
Partno          I don't know;
Date            07/26/88;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        ASSY;
Location        LOC;
/*************************************************************************/
/* This device generates the chip select signals for common memory.      */
/*************************************************************/
/* Allowable target device types: PAL16L8                                */
/*************************************************************/
/** Inputs **/
Pin 3           = ba0           ; /* CPU addresss bus                    */
Pin 4           = !baube        ; /* Upper bus enable                    */
Pin [5..8]      = [ba16..19]    ; /* CPU address bus                     */
/** Outputs **/
Pin [19..12]    = ![bacs1..8]   ; /* Common memory chip selects          */
/** Declarations and intermediate variable definitions **/
/** Logic equations **/
!bacs1=!(ba19 & !ba18 & !ba17 & !ba16 & !ba0);       /* Low byte of 8-  */
!bacs2=!(ba19 & !ba18 & !ba17 & !ba16 & !(!baube));  /*H. byte of 8-    */
!bacs3=!(ba19 & !ba18 & !ba17 & ba16 & !ba0);        /* Low byte of 9-  */
!bacs4=!(ba19 & !ba18 & !ba17 & ba16 & !(!baube));   /*High byte of 9   */
!bacs5=!(ba19 & !ba18 & ba17 & !ba16 & !ba0);        /* Low byte of A-  */
!bacs6=!(ba19 & !ba18 & ba17 & !ba16 & !(!baube));   /*High b. of A-*   */
!bacs7=!(ba19 & !ba18 & ba17 & ba16 & !ba0);         /* Low byte of B-  */
!bacs8=!(ba19 & !ba18 & ba17 & ba16 & !(!baube));    /*High byte of B-  */
```

Table 4.8

Logic Equations of PAL Device for Common Memory (Bank B) Chip Selects

```
Name            V50_CMD2;
Partno          I don't know;
Date            08/04/90;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        ASSY;
Location        LOC;
/*************************************************************************/
/* This device generates the chip select signals for common memory.      */
/*************************************************************************/
/* Allowable target device types: PAL16L8                                */
/*************************************************************************/
/** Inputs **/
Pin 3           = ab0           ; /* CPU addresss bus                    */
Pin 4           = !abube        ; /* Upper bus enable                    */
Pin [5..8]      = [ab16..19]    ; /* CPU address bus                     */
/** Outputs **/
Pin [19..12]    = ![abcs1..8]   ; /* Common memory chip selects          */
/** Declarations and intermediate variable definitions **/
/** Logic equations **/
!abcs1=!(ab19 & ab18 & !ab17 & !ab16 & !ab0);       /* Low byte of C-  */
!abcs2=!(ab19 & ab18 & !ab17 & !ab16 & !(!abube));  /*H. B. of C-      */
!abcs3=!(ab19 & ab18 & !ab17 & ab16 & !ab0);        /* Low byte of D-  */
!abcs4=!(ab19 & ab18 & !ab17 & ab16 & !(!abube));   /*H. B. of D-      */
!abcs5=!(ab19 & ab18 & ab17 & !ab16 & !ab0);        /* Low byte of E-  */
!abcs6=!(ab19 & ab18 & ab17 & !ab16 & !(!abube));   /*H. B. of E-      */
!abcs7=!(ab19 & ab18 & ab17 & ab16 & !ab0);         /* Low byte of F-  */
!abcs8=!(ab19 & ab18 & ab17 & ab16 & !(!abube));    /*H. B. of F-      */
```

4.4.2 Common ROM

This common ROM (CROM) is for starting-up the whole system. The monitor program is burned in. When the power is on or the reset switch is pressed, all SMPU's and SUBC inside the same stage, firstly, access this CROM, and copy the monitor program from CROM into private local memory (private memory) and start the monitor program from their own local memory. As soon as the system enters the steady states, CROM is disconnected from the common bus.

During the CROM is active, the common image memory is kept at passive state. This can be understood from Fig.4.2.G. When the power is on, $\overline{ET}$ is "H", an active "L" pulse (which must be wider than 35 ns) is fed to $\overline{PR}$ of D-flip-flop, and $\overline{Q}$ output is set to "L". This enables the address latches (μPB8286D) of CROM, and lets $\overline{OECROML}$ and

$\overline{\text{OECROMH}}$ go to the data bus transceivers (μPB8286D). Then passive $\overline{\text{ET}}$ (logic "H") disables the 3-state gate G1 and G2. The outputs of 3-state gate G1 and G2 are pulled up at "H", this means that all the control signals for common image memory are set at "H" (passive), and hence the common image memory will not be accessed.

Control signals of common image memory are generated by PAL devices PAL16L8A (V50_BSD1) and PAL16L8A (V50_BSD2). Table 4.9 and table 4.10 give their logic equations and Fig.4.7 illustrates the CROM accessing timing.

Table 4.9
Logic Equations of PAL Device for Generating Control Signals of Bus Latches and Transceivers of Global Image Memory and CROM

```
Name            V50_BSD1;
Partno          I don't know;
Date            07/26/88;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        ASSY;
Location        LOC;
/*************************************************************************/
/* This device generates the the control signals of bus latches and      */
/* transceivers of common image memory and common ROM.                   */
/*************************************************************************/
/* Allowable target device types: PAL16L8                                */
/*************************************************************************/
/** Inputs **/
Pin [1,7,9]     = [a0,a19,a18]      ;     /* CPU addresss bus          */
Pin 2           = !ube              ;     /* Upper bus enable          */
Pin [3.4]       = ![mamb,mbma]      ;     /* Memory bank enable        */
Pin 5           = !et               ;     /* Common image enable       */
/** Outputs **/
Pin [19,18]     = ![oesal,oesah]    ;     /* A bank data bus enable    */
Pin [17,16]     = ![oesbl,oesbh]    ;     /* B bank data bus enable    */
Pin [13,12]     = ![oecroml,oecromh];     /*CROM data bus enabl        */
/** Declarations and intermediate variable definitions **/
a       = !(!(!a19) & !(!ube));
b       = !(!(!a19) & !a0);
/** Logic equations **/
!oesal=!(!(!mamb) & !b & !(!a18));
!oesah=!(!(!mamb) & !a & !(!a18));
!oesbl=!(!(!mbma) & !b & !a18);
!oesah=!(!(!mbma) & !a & !a18);
!oecroml=!(!b & !(!(!et)));
!oecromh=!(!a & !(!(!et)));
```

Table 4.10
Logic Equations for Generating Control Signals of Bus Latches and Transceivers of Global Image Memory in the Upper Neighbor Stage

```
Name            V50_BSD1;
Partno          I don't know;
Date            07/26/88;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        ASSY;
Location        LOC;
/*************************************************************/
/* This device generates the the control signals of bus latches and      */
/* transceivers of common image memory and common ROM.                   */
/*************************************************************/
/* Allowable target device types: PAL16L8                                */
/*************************************************************/
/** Inputs **/
Pin [1,7,9]     = [a0,a19,a18]      ;       /* CPU addresss bus          */
Pin 2           = !ube              ;       /* Upper bus enable          */
Pin [3.4]       = ![mamb,mbma]      ;       /* Memory bank enable        */
Pin 5           = !et               ;       /* Common image enable       */
/** Outputs **/
Pin [19,18]     = ![oesal,oesah]    ;       /* A bank data bus enable    */
Pin [17,16]     = ![oesbl,oesbh]    ;       /* B bank data bus enable    */
Pin [13,12]     = ![oecroml,oecromh];       /*CROM data bus enabl        */
/** Declarations and intermediate variable definitions **/
a       = !(!(!a19) & !(!ube));
b       = !(!(!a19) & !a0);
/** Logic equations **/
!oesal=!(!(!mamb) & !b & !(!a18));
!oesah=!(!(!mamb) & !a & !(!a18));
!oesbl=!(!(!mbma) & !b & !a18);
!oesah=!(!(!mbma) & !a & !a18);
!oecroml=!(!b & !(!(!et)));
!oecromh=!(!a & !(!(!et)));
```

4.5 Pin Assignments of KIT-TA2 Board

KIT-TA2 is the same size with KIT-TA1 board. There is a 100-pin edge connector on the right side and a 69-pin connector on the left side for 50-line flat cable. The top view side and the inverse side are numbered BR1 to BR50, and AR1 to AR50, for the connecter on the right, BL1 to BL69, and AL1 to AL69, for the connector on the left, when located it as in the location of Fig.4.8. The meanings of these pins are explained in Table 4.11.

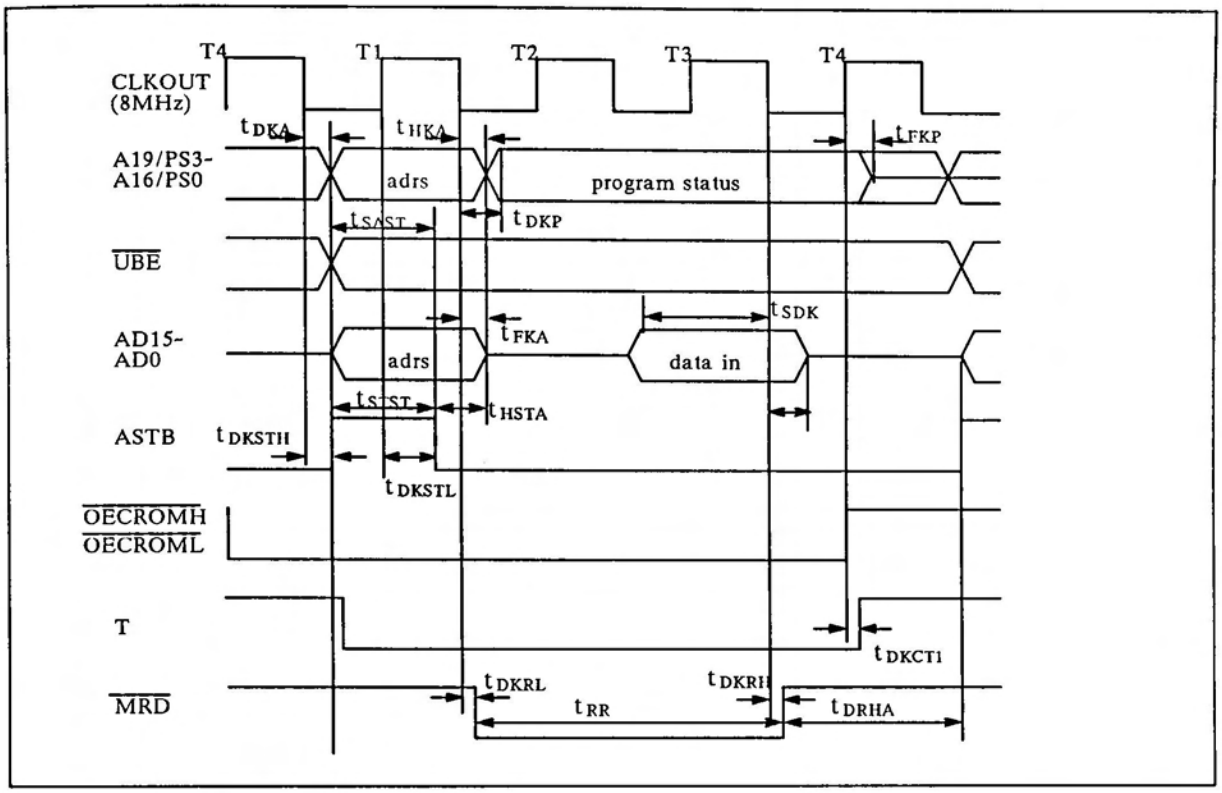

Fig.4.7. CROM read timing.

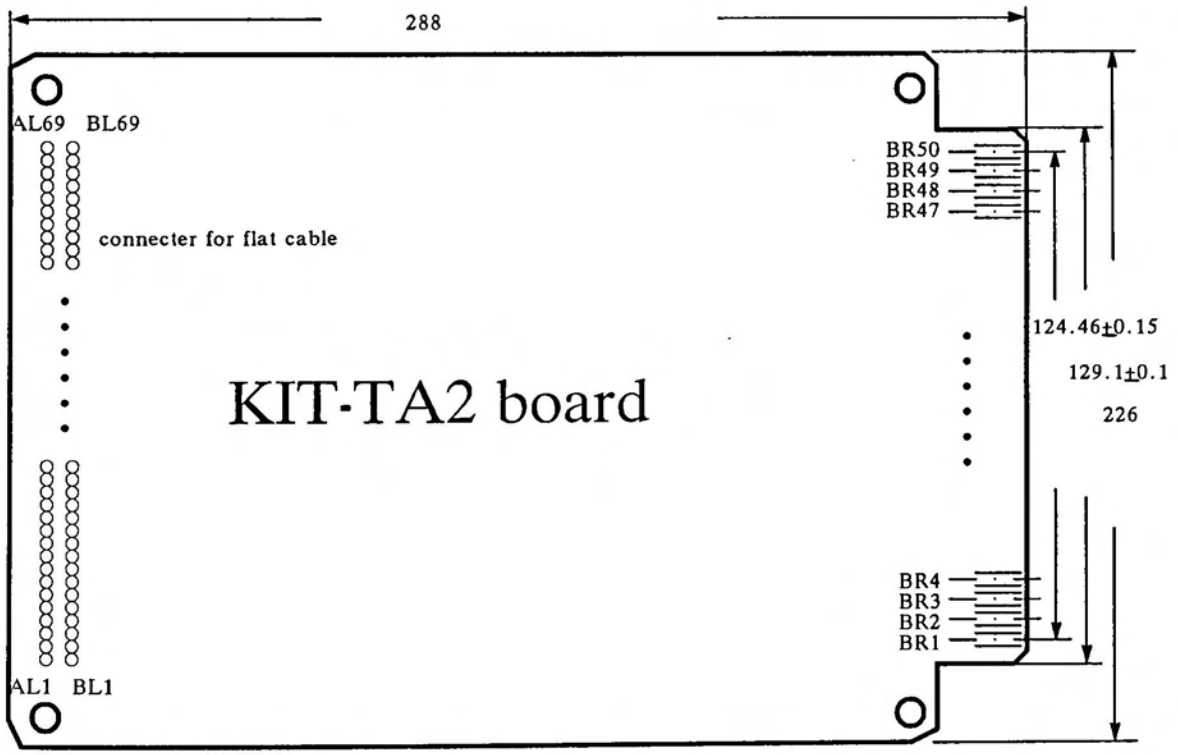

Fig.4.8. Outward appearance of KIT-TA2 board.

Table 4.11
Pin Assignments and Their Functions of KIT-TA2 Board

No.	SYBLS.	DIR.	FUNCTIONS
AR1	GND		
AR2	GND		
AR3	GND		
AR4	A9	3SO	Address bus
AR5	A10	3SO	Address bus
AR6	A11	3SO	Address bus
AR7	A12	3SO	Address bus
AR8	A13	3SO	Address bus
AR9	A14	3SO	Address bus
AR10	A15	3SO	Address bus
AR11	A16	3SO	Address bus
AR12	A17	3SO	Address bus
AR13	A19	3SO	Address bus
AR14	!ET	I	
AR15	!MAMB	O	Adrs latches enable of
AR16	!MBMA	O	upper stage
AR17	S3B		
AR18			
AR19	!BPRN7	O	Bus priority out
AR20	!BPRN6	O	Bus priority out
AR21	!BPRN5	O	Bus priority out
AR22	!BPRN4	O	Bus priority out
AR23	!BPRN3	O	Bus priority out
AR24	!BPRN2	O	Bus priority out
AR25	!BPRN1	O	Bus priority out
AR26	!BPRN0	O	Bus priority out
AR27	D0	I/O	Data bus
AR28	D1	I/O	Data bus
AR29	D2	I/O	Data bus
AR30	D3	I/O	Data bus
AR31	D4	I/O	Data bus
AR32	D5	I/O	Data bus
AR33	D6	I/O	Data bus
AR34	D7	I/O	Data bus
AR35	A0	3SO	Address bus
AR36			
AR37			
AR38			
AR39			
AR40	RESET	I	Active high reset
AR41	!MBMA	I	Bus switch enable
AR42	!MAMB	I	Bus switch enable
AR43	!OEAUH	O	Data bus transceiver
AR44	!OEAUL	O	enable of upper stage
AR45	!OEBUH	O	Data bus transceiver
AR46	!OEBUL	O	enable of upper stage
AR47	S1B		
AR48	VCC		
AR49	VCC		
AR50	VCC		

No.	SYBLS.	DIR.	FUNCTIONS
BR1	GND		
BR2	GND		
BR3	GND		
BR4	A1	3SO	Address bus
BR5	A2	3SO	Address bus
BR6	A3	3SO	Address bus
BR7	A4	3SO	Address bus
BR8	A5	3SO	Address bus
BR9	A6	3SO	Address bus
BR10	A7	3SO	Address bus
BR11	A8	3SO	Address bus
BR12	A18	3SO	Address bus
BR13	S1A		
BR14	S2A		
BR15	S3A		
BR16	S4A		
BR17	S5A		
BR18	NMI		
BR19	!BREQ	I	Common bus request
BR20	!BREQ	I	Common bus request
BR21	!BREQ	I	Common bus request
BR22	!BREQ	I	Common bus request
BR23	!BREQ	I	Common bus request
BR24	!BREQ	I	Common bus request
BR25	!BREQ	I	Common bus request
BR26	!BREQ	I	Common bus request
BR27	D8	I/O	Data bus
BR28	D9	I/O	Data bus
BR29	D10	I/O	Data bus
BR30	D11	I/O	Data bus
BR31	D12	I/O	Data bus
BR32	D13	I/O	Data bus
BR33	D14	I/O	Data bus
BR34	D15	I/O	Data bus
BR35	!UBE	3SO	Upper byte enable
BR36			
BR37			
BR38	CLK	I	CPU clock
BR39	BCLK	I	Bus clock
BR40	!RESET	I	Active low reset
BR41	!WR	3SO	Memory write enable
BR42	!RD	3SO	Memory read enable
BR43	DT!/R	3SO	Buffer write/read
BR44			
BR45	!CBREQ	I/O	Wired 'or' of all !CBREQ
BR46	!BUSY	I/O	Wired 'or' of all !busy
BR47	S2B		
BR48	VCC		
BR49	VCC		
BR50	VCC		

to be continued

continue

No.	SYBLS.	DIR.	FUNCTIONS
AL1	GND		
AL2	GND		
AL3	GND		
AL4	GND		
AL5	GND		
AL6	GND		
AL7	GND		
AL8	GND		
AL9	GND		
AL10	GND		
AL11	GND		
AL12	GND		
AL13	GND		
AL14	GND		
AL15	GND		
AL16	GND		
AL17	GND		
AL18	GND		
AL19	GND		
AL20	GND		
AL21	GND		
AL22	GND		
AL23	GND		
AL24	GND		
AL25	GND		
AL26	GND		
AL27	GND		
AL28	GND		
AL29	GND		
AL30-39	GND		
AL40	GND		
AL41	GND		
AL42	GND		
AL43	GND		
AL44	GND		
AL45	GND		
AL46	GND		
AL47	GND		
AL48	GND		
AL49	GND		
AL50	GND		
AL51	GND		
AL52	GND		
AL53	GND		
AL54	GND		
AL55	GND		
AL56	GND		
AL57	GND		
AL58	GND		
AL59	GND		
AL60	GND		
AL61	GND		
AL62	GND		
AL63-69	GND		

No.	SYBLS.	DIR.	FUNCTIONS
BL1	GND		
BL2	GND		
BL3	GND		
BL4	GND		
BL5	A1	3SO	Address bus
BL6	A2	3SO	Address bus
BL7	A3	3SO	Address bus
BL8	A4	3SO	Address bus
BL9	A5	3SO	Address bus
BL10	A6	3SO	Address bus
BL11	A7	3SO	Address bus
BL12	A8	3SO	Address bus
BL13	A9	3SO	Address bus
BL14	A10	3SO	Address bus
BL15	A11	3SO	Address bus
BL16	A12	3SO	Address bus
BL17	A13	3SO	Address bus
BL18	A14	3SO	Address bus
BL19	A15	3SO	Address bus
BL20	A16	3SO	Address bus
BL21	A17	3SO	Address bus
BL22	A18	3SO	Address bus
BL23	A19	3SO	Address bus
BL24	D0	I/O	Data bus
BL25	D8	I/O	Data bus
BL26	D1	I/O	Data bus
BL27	D9	I/O	Data bus
BL28	D2	I/O	Data bus
BL29	D10	I/O	Data bus
BL30-39	GND		
BL40	D3	I/O	Data bus
BL41	D11	I/O	Data bus
BL42	D4	I/O	Data bus
BL43	D12	I/O	Data bus
BL44	D5	I/O	Data bus
BL45	D13	I/O	Data bus
BL46	D6	I/O	Data bus
BL47	D14	I/O	Data bus
BL48	D7	I/O	Data bus
BL49	D15	I/O	Data bus
BL50	!UBE	3SO	Upper byte enable
BL51	A0	3SO	Address bus
BL52			
BL53			
BL54	!WR	3SO	Memory write enable
BL55	!RD	3SO	Memory read enable
BL56	DTIR	3SO	Transceiver strobe
BL57	!MAMB	I	Adrs bus latches enable
BL58	!MBMA	I	Adrs bus latches enable
BL59	!OEAUH	I	Data bus transceiver enable
BL60	!OEAUL	I	Data bus transceiver enable
BL61	!OEBUH	I	Data bus transceiver enable
BL62	!OEBUL	I	Data bus transceiver enable
BL63-69	GND		

4.6 Automatic Synchronization among Stages

The PE's inside a stage of HIGIPS operate asynchronously. But to get the pipeline performance, stages of HIGIPS must be synchronous, i.e., the bus switches of every stage must be set *on/off* at the same time. This is performed by the circuit in Fig.4.2.H.

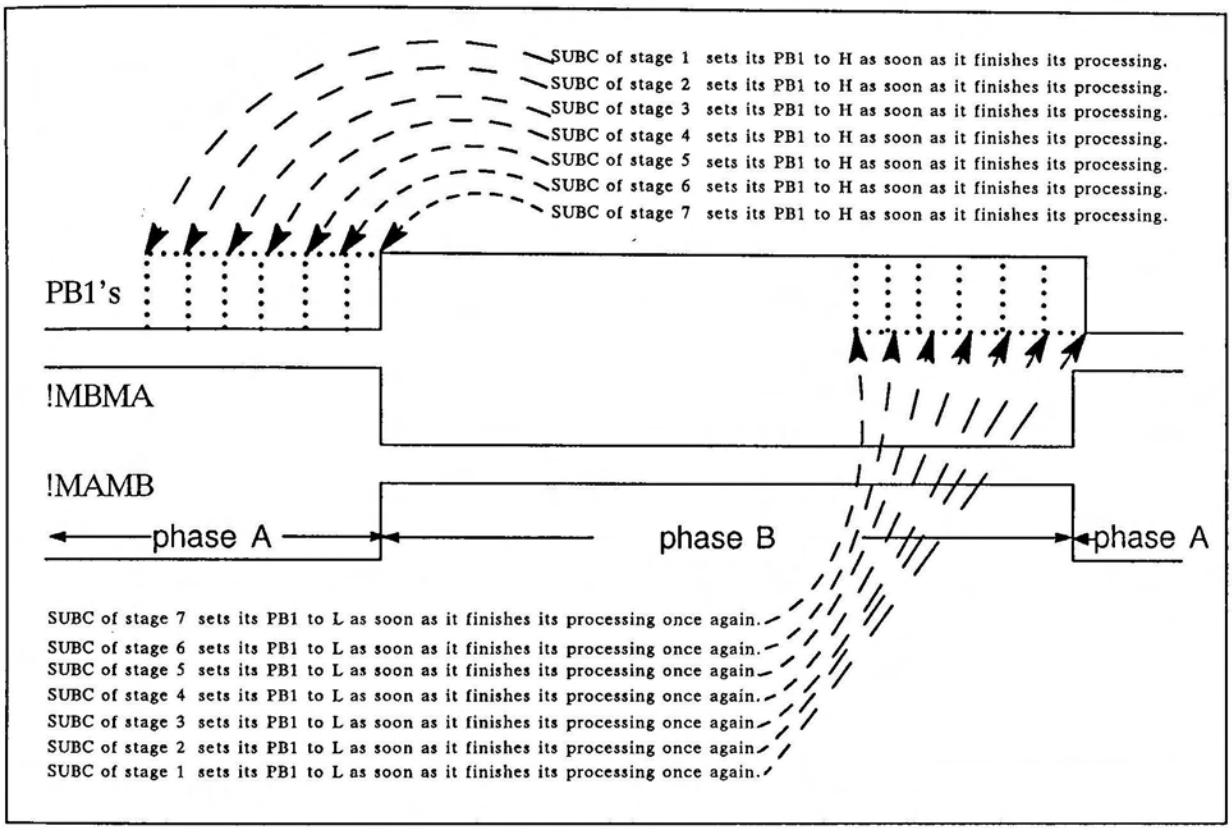

Fig.4.9. Principle of automatic synchronization between stages.

In the phase **A**, SUBC of each stage set its PB1 to "L", and $\overline{PR}$ is active "L" and $\overline{CLR}$ active "H", so the outputs $\overline{MAMB}$ and $\overline{MBMA}$ of D-flip-flop are "L" and "H", respectively. As soon as the processing of a stage is finished, the SUBC of it will set its PB1 to "H". When all SUBC's of HIGIPS set their PB1's to "H", the D-FF-1 alternates, and $\overline{MAMB}$ becomes "H" and $\overline{MBMA}$ boecomes "L", and the bus switches are turned into the phase **B**. Similarly, when all SUBC's set their PB1's to "L", the bus switches are turned back to the phase *A*. (see Fig.4.9). $\overline{CLR}$ and $\overline{PR}$ are generated by a PAL device PAL16L8A (V50_SW1). Table 4.12 gives its logic equations.

Table 4.12
Logic Equations for Generating Control Signals of
Automatic Stage Synchronization Circuit

```
Name            V50_SW1;
Partno          I don't know;
Date            08/20/90;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        ASSY;
Location        LOC;
/*************************************************************************/
/* This device generates control signals for !MAMB, !MBMA and !ET.       */
/*************************************************************************/
/* Allowable target device types: PAL16L8                                */
/*************************************************************************/
/** Inputs **/
Pin [1..7]          = ![sa1..sa7]        ; /* PA0 from SUBC of each stage   */
/** Outputs **/
Pin 12 = !pr                  ; /* !PR of D-flip-flpo                    */
Pin 13 = !clr                 ; /* !CLR of D-flip-flop                   */
/** Declarations and intermediate variable definitions **/
/** Logic equations **/
!pr=!(!(!sa1) & !(!sa2) & !(!sa3) & !(!sa4) & !(!sa5) & !(!sa6) & !(!sa7));
!clr=!(!sa1 & !sa2 & !sa3 & !sa4 & !sa5 & !sa6 & !sa7);
```

4.7 Automatic Restarting of the Whole System

Once a stage finishes its processing, it gets into halting state. When all stages complete their processing, the bus switches are turned into another phase, and all stages start the same processing program but deal with different image data (i.e., PM's are connected to different memory banks). To restart HIGIPS, an active "H" signal has to be fed to NMI-pin of SUBC CPU in each stage. The circuit to generate this interrupt signal is shown in Fig.4.2.I.

SUBC of a stage sets its PB0 to "L" at the end of processing, and enter the halting state. When every SUBC of HIGIPS sets its PB0 to "L", NMI becomes active "H". The NMI signal is sent to SUBC of every stage. This restarts PM of every stage. The principle of it is shown in Fig.4.10. NMI is generated by a PAL device PAL16L8A (V50_INT), and table 4.13 gives its logic equations.

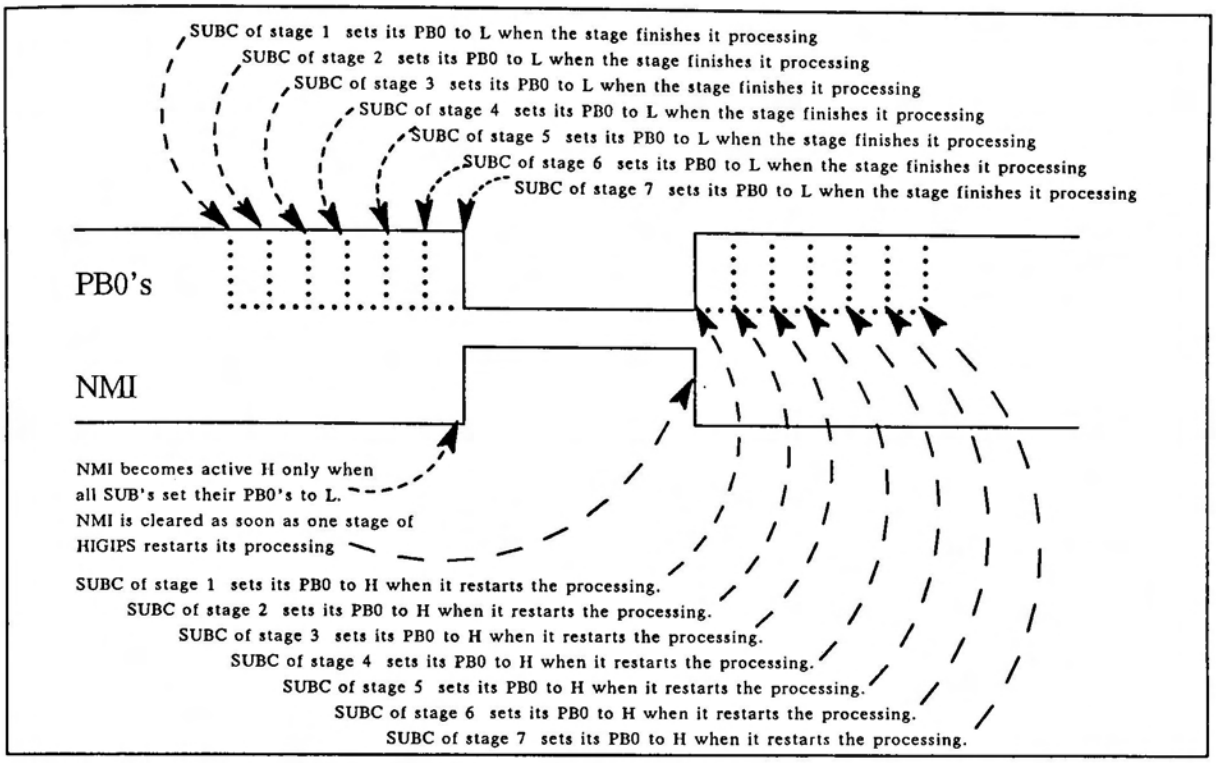

Fig.4.10. Principle of automatic restarting circuit.

Table 4.13

Logic Equations for Control Signals of Automatic Restarting of the Whole System

```
Name            V50_INT;
Partno          I don't know;
Date            08/20/90;
Revision        01;
Designer        F.H. Yao;
Company         K.I.T.;
Assembly        ASSY;
Location        LOC;
/*************************************************************************/
/* This device generates control signals for !MAMB, !MBMA and !ET.       */
/*************************************************************************/
/* Allowable target device types: PAL16L8                                */
/*************************************************************************/
/** Inputs **/
Pin [1..7]          = ![sb1..sb7]        ; /* PB0 from SUBC of each stage   */
/** Outputs **/
Pin 12 = !rs                    ; /* !Input of RS-flip-flpo                */
/** Declarations and intermediate variable definitions **/
/** Logic equations **/
!rs=!(!(!sb1) & !(!sb2) & !(!sb3) & !(!sb4) & !(!sb5) & !(!sb6) & !(!sb7));
```

4.8 Picture Input Unit

The PIU is composed of one KIT-TA1 board as PIC equipped with the parallel I/O interface, one KIT-TA2 board as MM, and the HyPER VISION board as ADM [109, 123] which is assigned to I/O port with address of 8D0H-8D7H. The HyPER VISION board digitizes the standard NTSC or Y/C separate NTSC signal and places it into MM.

Note here that the HyPER VISION board is the extended board developed for the personal computer (which is the PC98 series product of NEC) by Digital Arts Ltd..

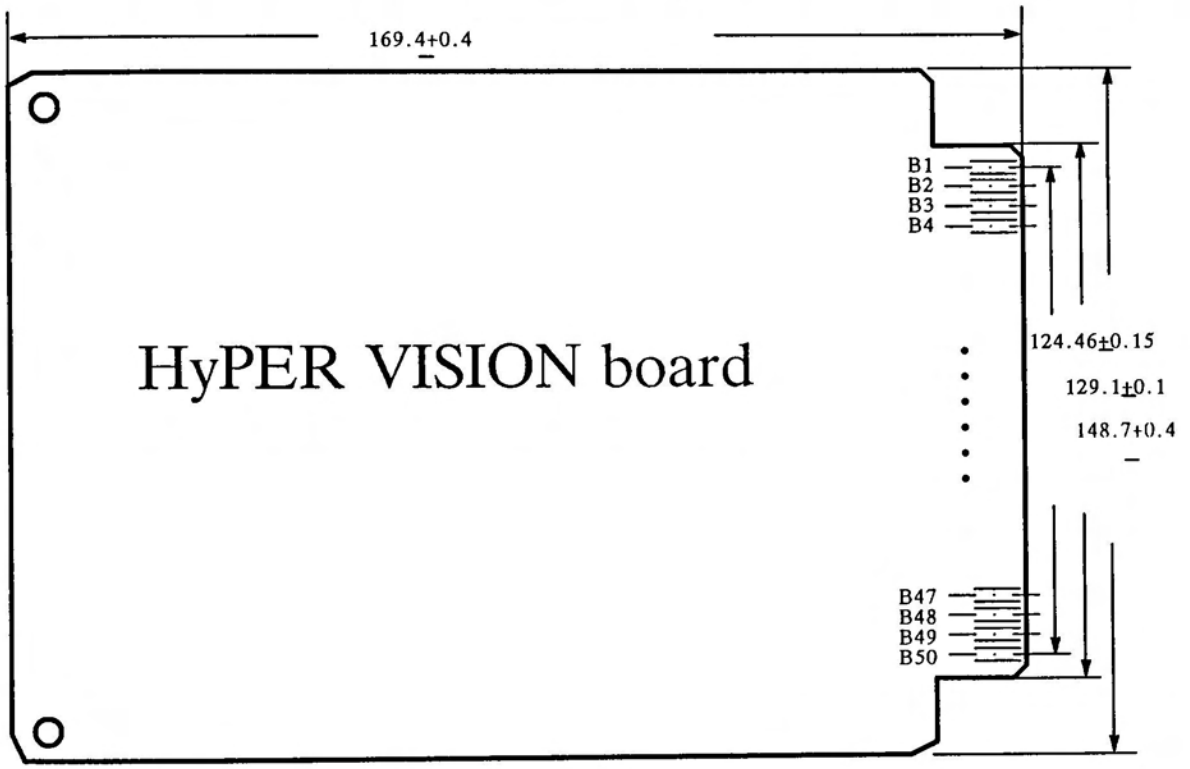

Fig.4.11. Outward appearance of HyPER VISION board.

Since KIT-TA1 board is designed for maximal mode system, it can not be used as the controller of PIU directly. It needs some slight changes, i.e., address bus, data bus and control bus must be from the local bus of PE (see Fig.4.12). Furthermore, because HyPER VISION board was developed for NEC PC98 series personal computer and the pin assignments are different with KIT-TA1 board, an interface between them is needed. Table 4.14 gives the pin assignments of HyPER VISION board when located it as in Fig.4.11, numbered from B1 to B50 and A1 to A50 for the top view side and inverse side, respectively. Fig.4.12 shows the construction of the interface board between PIC and HyPER VISION board. The PIU has the following features:

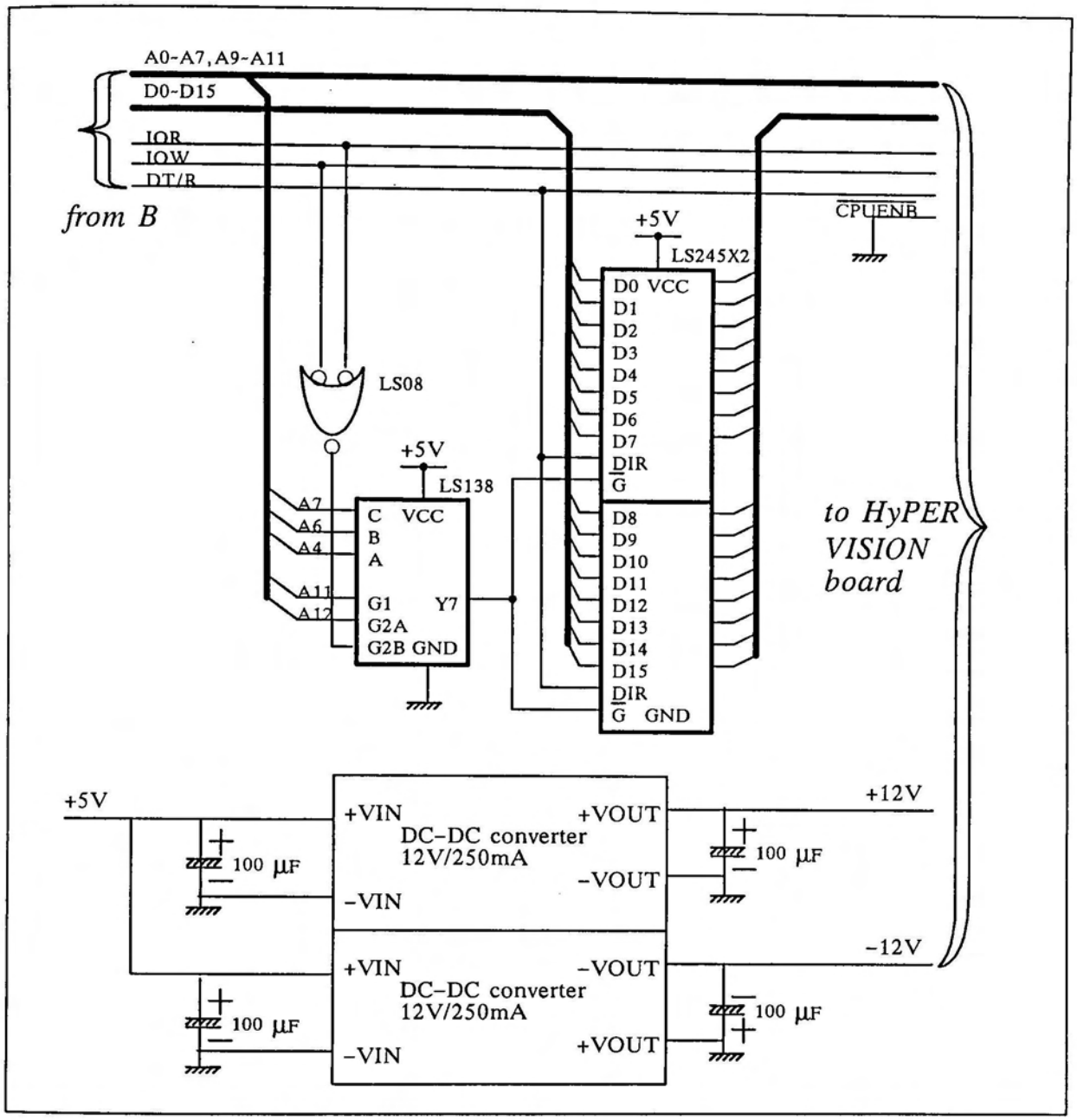

Fig.4.12. Interface circuit between PIC and ADM.

(1) 8-bit digitizing levels for black-white visual signal and 8-bit digitizing level for R, G, and B components of color visual signal with sampling frequency of 4×fsc (14.32-MHz);

(2) 640H×400V resolution which is the same with that of the display device of NEC PC98 series personal computers;

(3) being able to handle both the standard NTSC visual signal and pseudo visual signal, e.g., the output of VTR;

(4) corresponding to both the composite video signal and Y/C separate video signal.

Table 4.14
Pin Assignments and Their Functions of HyPER VISION Board

No.	SYBLS.	DIR.	FUNCTIONS
A1	GND		
A2			
A3			
A4	A0	3SO	Address bus
A5	A1	3SO	Address bus
A6	A2	3SO	Address bus
A7	A3	3SO	Address bus
A8	A4	3SO	Address bus
A9	A5	3SO	Address bus
A10	A6	3SO	Address bus
A11	GND		Address bus
A12	A7	3SO	Address bus
A13			
A14	A9	3SO	Address bus
A15	A10	3SO	Address bus
A16	A11	3SO	Address bus
A17			
A18			
A19			
A20			
A21	GND		
A22			
A23			
A24			
A25			
A26			
A27			
A28			
A29			
A30			
A31	GND		
A32			
A33	!IOR	3SO	I/O read enable
A34	!IOW	3SO	I/O write enable
A35			
A36			
A37			
A38			
A39			
A40			
A41	GND		
A42	CPUENB	I	CPU is using common bus
A43			
A44	UBE	I	Upper byte enable
A45			
A46			
A47			
A48			
A49	VCC		
A50	VCC		

No.	SYBLS.	DIR.	FUNCTIONS
B1	GND		
B2			
B3			
B4	D0	I/O	Data bus
B5	D1	I/O	Data bus
B6	D2	I/O	Data bus
B7	D3	I/O	Data bus
B8	D4	I/O	Data bus
B9	D5	I/O	Data bus
B10	D6	I/O	Data bus
B11	GND		
B12	D7	I/O	Data bus
B13	D8	I/O	Data bus
B14	D9	I/O	Data bus
B15	D10	I/O	Data bus
B16	D11	I/O	Data bus
B17	D12	I/O	Data bus
B18	D13	I/O	Data bus
B19	D14	I/O	Data bus
B20	D15	I/O	Data bus
B21	GND		
B22	+12V		
B23	+12V		
B24			
B25			
B26			
B27			
B28			
B29			
B30			
B31	GND		
B32	-12V		
B33	-12V		
B34	!RESET	I	Reset from PIC
B35			
B36			
B37			
B38			
B39			
B40			
B41	GND		
B42			
B43			
B44			
B45			
B46			
B47			
B48			
B49	VCC		
B50	VCC		

Note: Blank means no touch with these pins.

4.9 Picture Output Unit

The POU comprises one KIT-TA1 board set with the parallel I/O interface (GP-IB) as POC, one KIT-TA2 board as MM, the HyPER FRAME board as FBM, and the analog RGB display to output result.

FBM is composed of 768 kilobytes dual port image memory (256 kilobytes for R, G, and B data, respectively). It is assigned to the I/O port with address of 00D0H-00EEH. The image data can be written/read to/from the frame buffer memory.

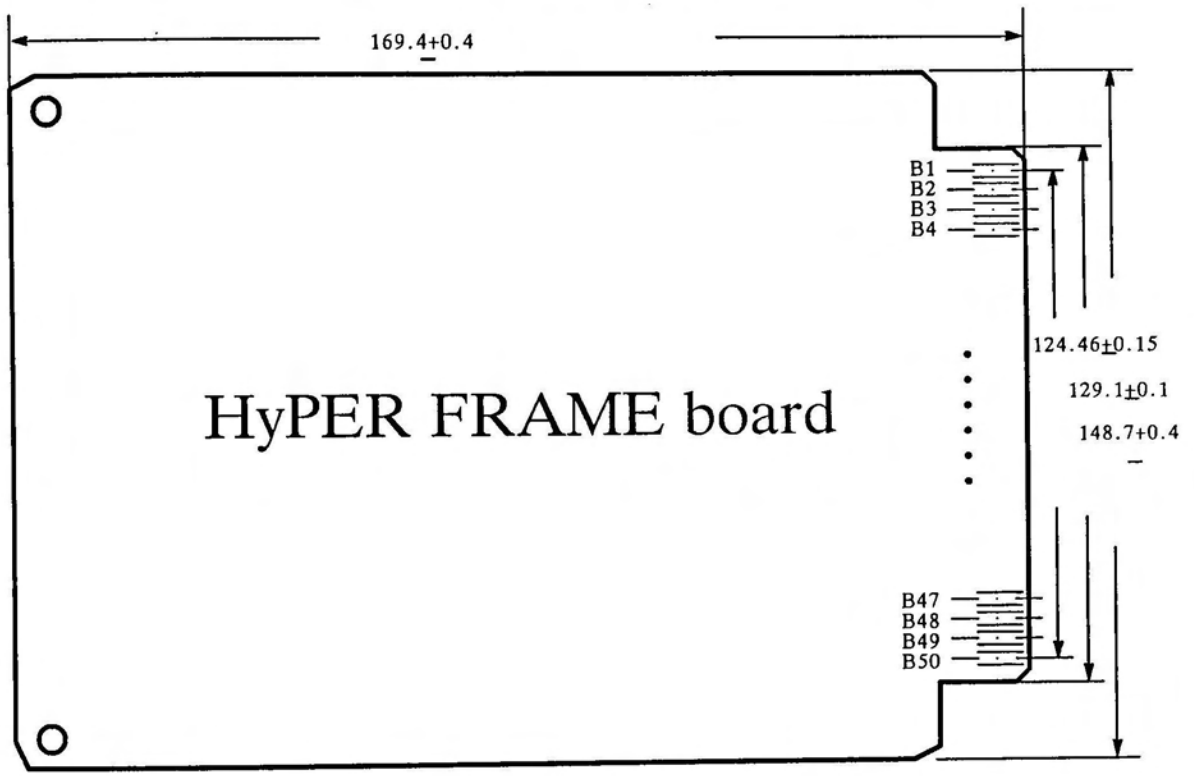

Fig.4.13. Outward appearance of HyPER FRAME board.

Note here that the HyPER FRAME board is the extended board developed for the personal computer (which is the PC98 series product of NEC) by Digital Arts Ltd.. Its outward appearance is shown in Fig.4.13, and table 4.15 gives its pin assignments and their meanings (with the location as in Fig.4.13, and numbering the top view side and inverse side from B1 to B50, and A1 to A50, respectively).

KIT-TA1 has also to be changed slightly to take the role of POC. And also, a interface between POC and FBM is needed (see Fig.4.14). The POU has the following features:

(1) 768-kilobyte image memory (256-kilobyte for each of R, G and B signal);
(2) 640H×400V resolution which is the same with that of display device of NEC PC98 personal computer;

(3) 256×256×256 (24-bit) kinds of color;
(4) Changeable quantum bits: 8-bit (full color), 4-bit (4096 kinds of color), 2-bit (64 kinds of color), and 1-bit (8 kinds of color);
(5) 1, 2, 4, and 8 time(s) zoom at x- or y-direction, independently;
(6) Smooth scroll at x- or y-direction, independently.

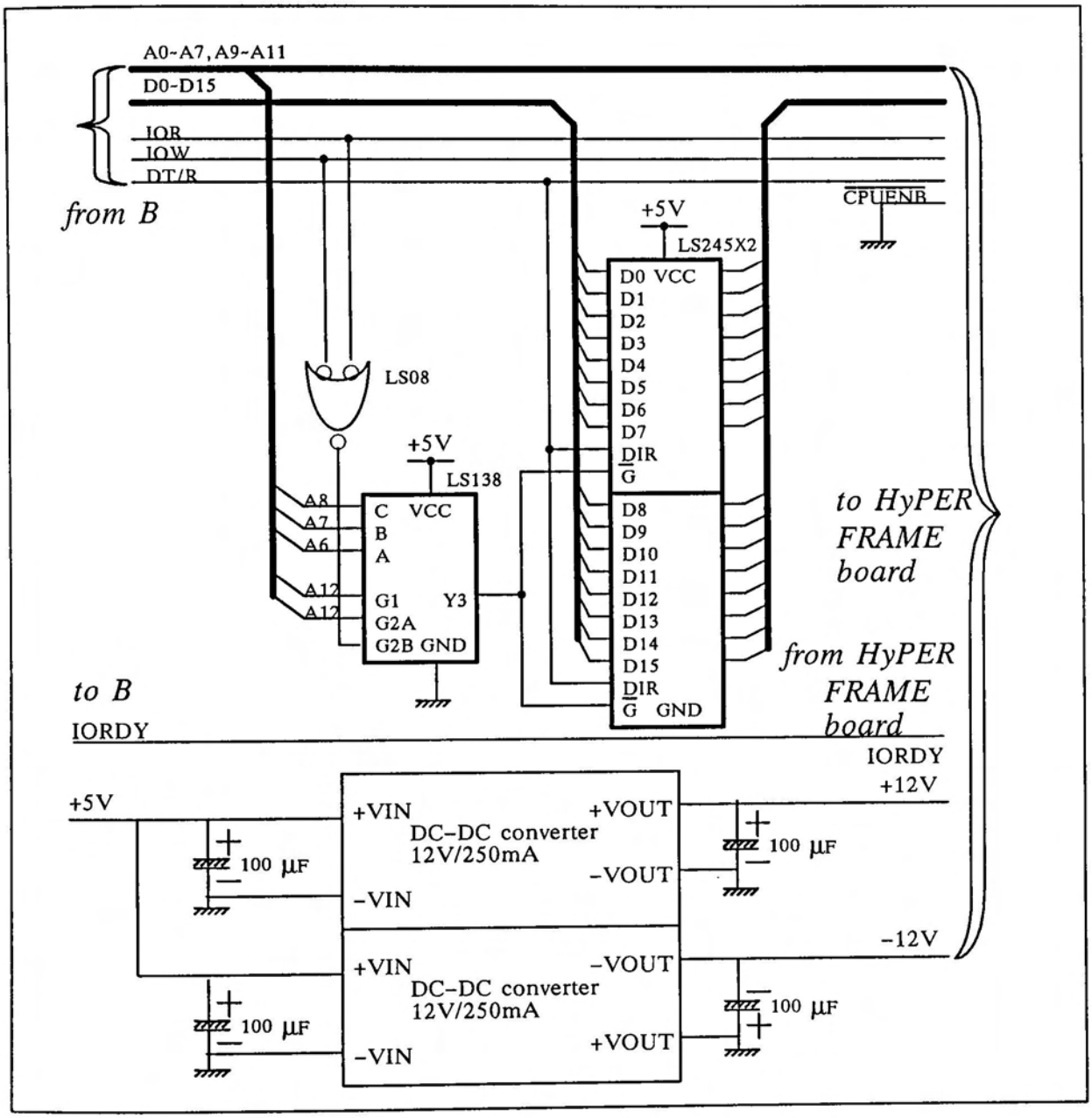

Fig.4.14. Interface between POC and FBM.

Table 4.15

Pin Assignments and Their Functions of HyPER FRAME Board

No.	SYBLS.	DIR.	FUNCTIONS
A1	GND		
A2			
A3			
A4	A0	3SO	Address bus
A5	A1	3SO	Address bus
A6	A2	3SO	Address bus
A7	A3	3SO	Address bus
A8	A4	3SO	Address bus
A9	A5	3SO	Address bus
A10	A6	3SO	Address bus
A11	GND		Address bus
A12	A7	3SO	Address bus
A13			
A14	A9	3SO	Address bus
A15	A10	3SO	Address bus
A16	A11	3SO	Address bus
A17			
A18			
A19			
A20			
A21	GND		
A22			
A23			
A24			
A25			
A26			
A27			
A28			
A29			
A30			
A31	GND		
A32			
A33	!IOR	3SO	I/O read enable
A34	!IOW	3SO	I/O write enable
A35			
A36			
A37			
A38			
A39			
A40			
A41	GND		
A42	CPUENB	I	CPU is using common bus
A43			
A44	UBE	I	Upper byte enable
A45			
A46			
A47			
A48			
A49	VCC		
A50	VCC		

No.	SYBLS.	DIR.	FUNCTIONS
B1	GND		
B2			
B3			
B4	D0	I/O	Data bus
B5	D1	I/O	Data bus
B6	D2	I/O	Data bus
B7	D3	I/O	Data bus
B8	D4	I/O	Data bus
B9	D5	I/O	Data bus
B10	D6	I/O	Data bus
B11	GND		
B12	D7	I/O	Data bus
B13	D8	I/O	Data bus
B14	D9	I/O	Data bus
B15	D10	I/O	Data bus
B16	D11	I/O	Data bus
B17	D12	I/O	Data bus
B18	D13	I/O	Data bus
B19	D14	I/O	Data bus
B20	D15	I/O	Data bus
B21	GND		
B22	+12V		
B23	+12V		
B24			
B25			
B26			
B27			
B28			
B29			
B30			
B31	GND		
B32	-12V		
B33	-12V		
B34	!RESET	I	Reset from PIC
B35			
B36			
B37			
B38			
B39			
B40			
B41	GND		
B42			
B43			
B44			
B45			
B46			
B47			
B48			
B49	VCC		
B50	VCC		

Note: Blank means no touch with these pins.

4.10 Outward Appearance of Prototype HIGIPS

A prototype HIGIPS which has three processing stages and each stage of it has three processors (but it can be easily extended to eight) is developed, and is completely working.

Photo 4.1, 4.2 and 4.3 show the PIC board, PIC/ADM interface board and ADM board respectively. Photo 4.4, and 4.5 show the KIT-TA1 and KIT-TA2 board accordingly. Photo 4.6, 4.7 and 4.8 show the POC board, POC/FBM interface board and FBM board, separately. And photo 4.9 shows the whole prototype HIGIPS system.

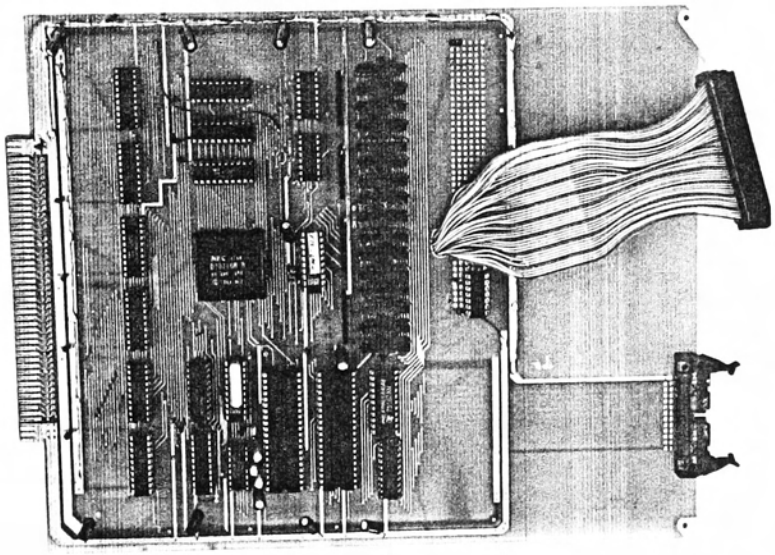

Photo 4.1. PIC board.

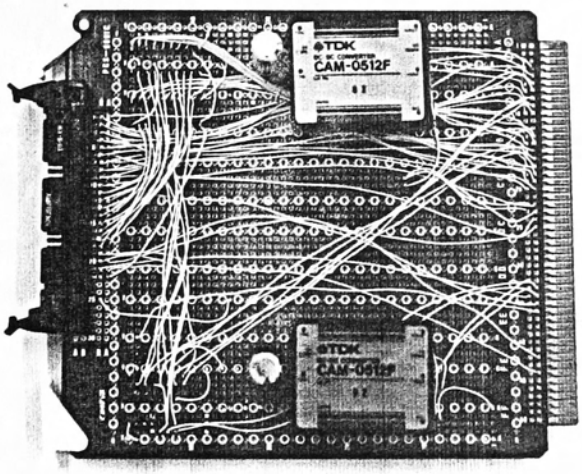

Photo 4.2. PIC/ADM interface board.

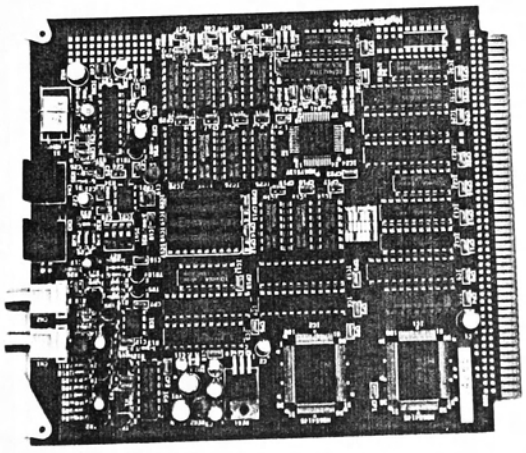

Photo 4.3. ADM board.

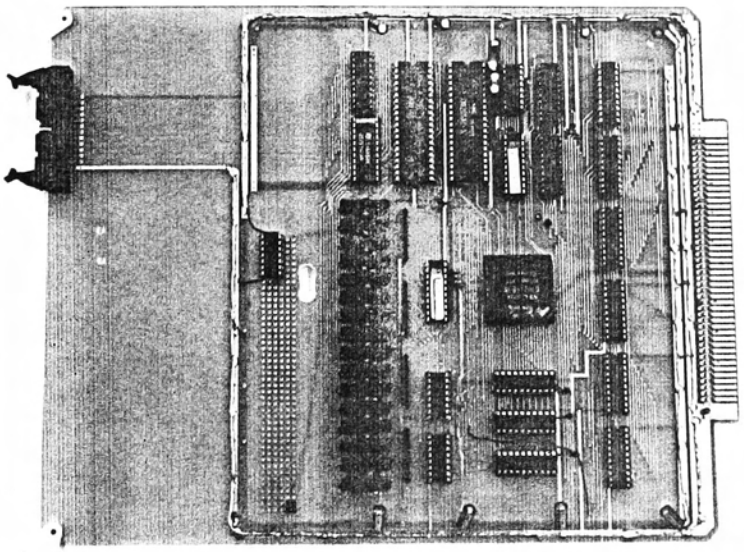

Photo 4.4. KIT-TA1 board.

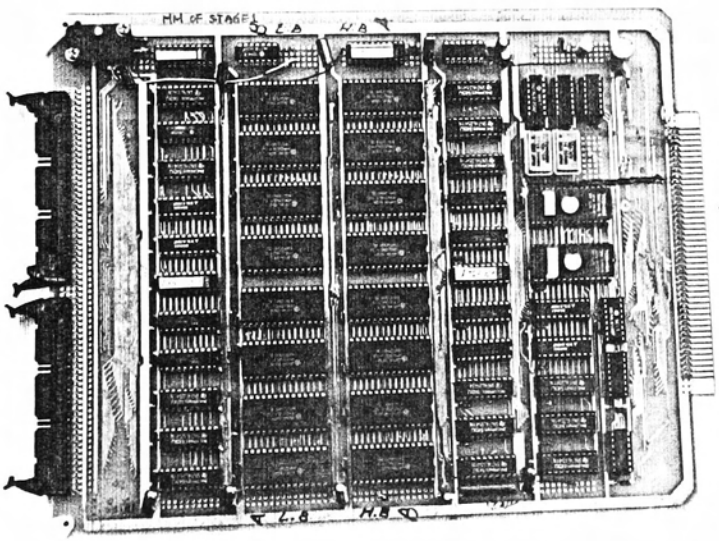

Photo 4.5. KIT-TA2 board.

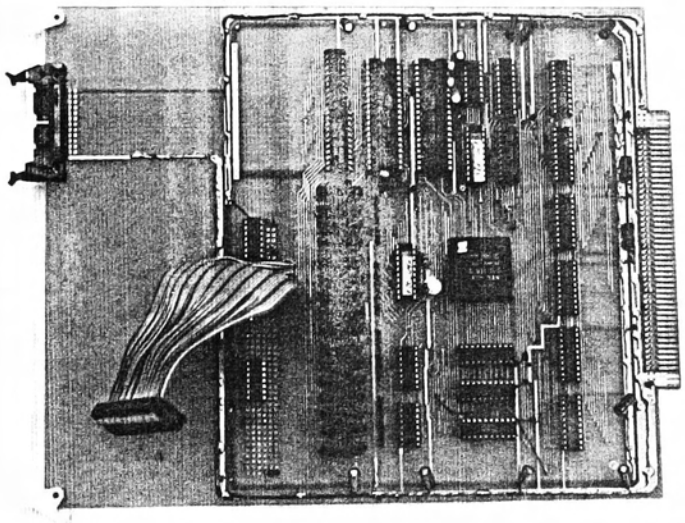

Photo 4.6. POC board.

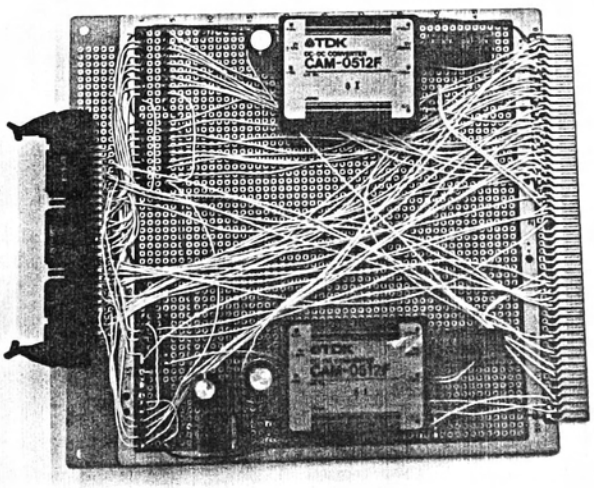

Photo 4.7. POC/FBM interface board.

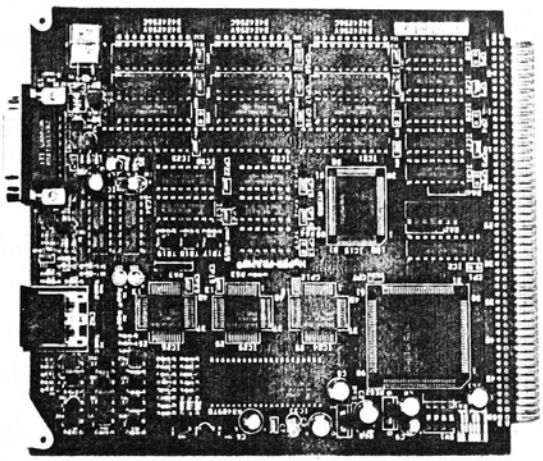

Photo 4.8. FBM board.

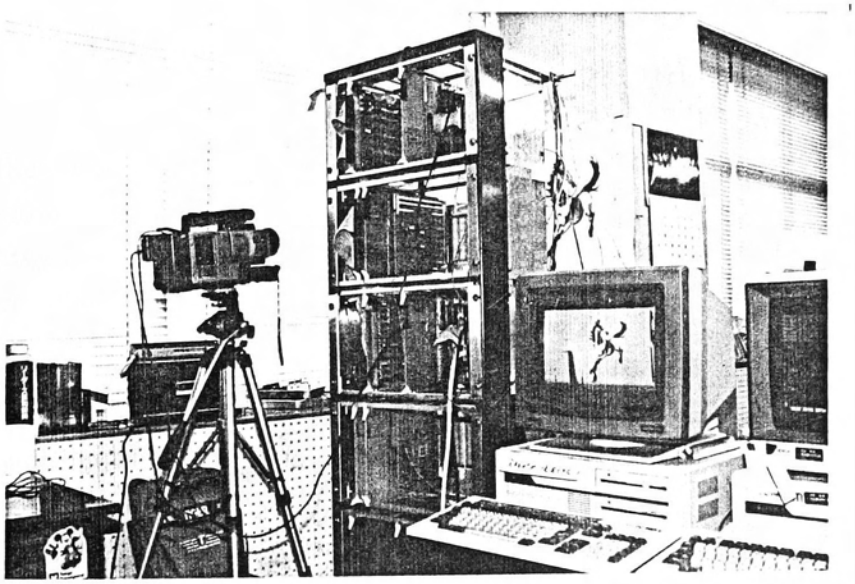

Photo 4.9. Whole prototype HIGIPS system.

Chapter 5

HIGIPS SOFTWARE STRATEGIES

5.1 A Brief Background Review of Related Work

A large number of languages have been coded to handle parallel array processors. These have tended to be at the assembly language level, or to be subroutine calls in FORTRAN. But they include a number of languages that are quite simple and straightforward to use, and offer the user great power (to a large extent because the parallel arrays, with their very large numbers of processors, are so powerful). Most of these languages execute entirely on a particular machine with parallel hardware (or on a simulation of parallel hardware), and are directed toward image processing.

Gudmundsson and Kruse (Linkoping University, Sweden) [35] have developed a very nice ALGOL-like language in which programs can be coded that call both the PICAP parallel processor and the serial host computer. It contains two major parallel constructs, for (i) logical, and (ii) numerical operations, with the format:

```
(type)    1 2 3
          4 5 6    = result
          7 8 9
```

That is, the programmer is asked to specify a 3×3 array of nine values (these may be boolean values, or integers specifying weights or inequalities), plus an operation and a result. This closely reflects the nearest-neighbor structure of the PICAP-1 array (and of most other of today's hardware arrays.

Uhr (University of Wisconsin, USA) [97] has developed a higher-level language (also called PascalPL) for the CLIP parallel array that makes use of "compounds" and "implications" similar to, and precursors of, some of the constructs in the PASCAL-based PascalPL.

Levialdi and his associates (Consiglio Nazionale delle Ricerche, Istituto di Cibernetica, Iatly) [55] have developed PIXAL, which consists of parallel extensions embeded in ALGOL-60 (which they are using chiefly because PASCAL is not available on their computer). PIXAL uses a "frame" construct that allows the programmer to specify a set of relative locations (not only nearest-neighbors) to be locked at everywhere (that is, relative to each cell in the array), and a "mask" construct with which the programmer can

specify a set of weights to be applied to the relative locations specified in the coordinate structure.

Douglass (University of Virginia, USA) [18] has proposed extensions to PASCAL to handle parallel arrays, building on earlier proposed extensions by Pratt and Ison to handle networks. These include a "fork" operation to set up new processor that are executing different sets of instructions in parallel, and a "split" operation, to invoke whole sets (e.g., arrays) of processors to execute the same set of instructions, but each on different data (e.g., different local regions of a visual image). The programmer can specify "windows" (sets of (relative) coordinate locations).

Almost all of these languages use an operation on a set of relative locations as their key construct. The languages designed for a specific hardware array will build in the interconnection pattern of that array (usually, as in the case of PICAP-1, the 3×3 sub-array of the nearest neighbors). PIXAL's "mask" and Douglass' "window" generalize this to any set of arbitrarily distant neighbors. (But when such a language is actually executed on a hardware-parallel array, this gives excessively long sequences of nearest-neighbor shifting operations.) Uhr added constructs for the convenient compounding and implying of information over several different arrays. Douglass make suggestions for constructs to handle much more general networks, as well as SIMD arrays.

To shorten the developing time, HIGIPS attempts to utilize the most popular C high-level language to developing the processing program. This will be discussed in the next section.

5.2 Image Processing on HIGIPS

HIGIPS is designed and developed for high-level image processing and analysis. To perform the high-level image processing (which is usually complicated and handles not only image data but also a huge amounts of symbols, inferences equations), the low-level language (e.g., assembler language) is not suitable. Therefore, we plan to use the existing C language (which is the off-the-shelf software for NEC PC98 series personal computer under MS-DOS operating system). The procedures of image processing on HIGIPS are: 1) to develop the processing program on SC with C language and make the executable file; 2) to download this executable file to HIGIPS; and 3) to send a "RUN" message from SC to start the image processing.

5.2.1 Image Processing Flow Graph

For any image processing, we can get the corresponding image processing flow graph. An example of the flow graph is given in Fig.5.1. A node in the flow graph represents a

basic processing unit such as enhancement, filtering, thinning, edge detection, and template matching. Each processing unit is called a task. A branch in the flow graph represents the relationship between the processing units, e.g., the result of a processing unit is the input of another processing unit.

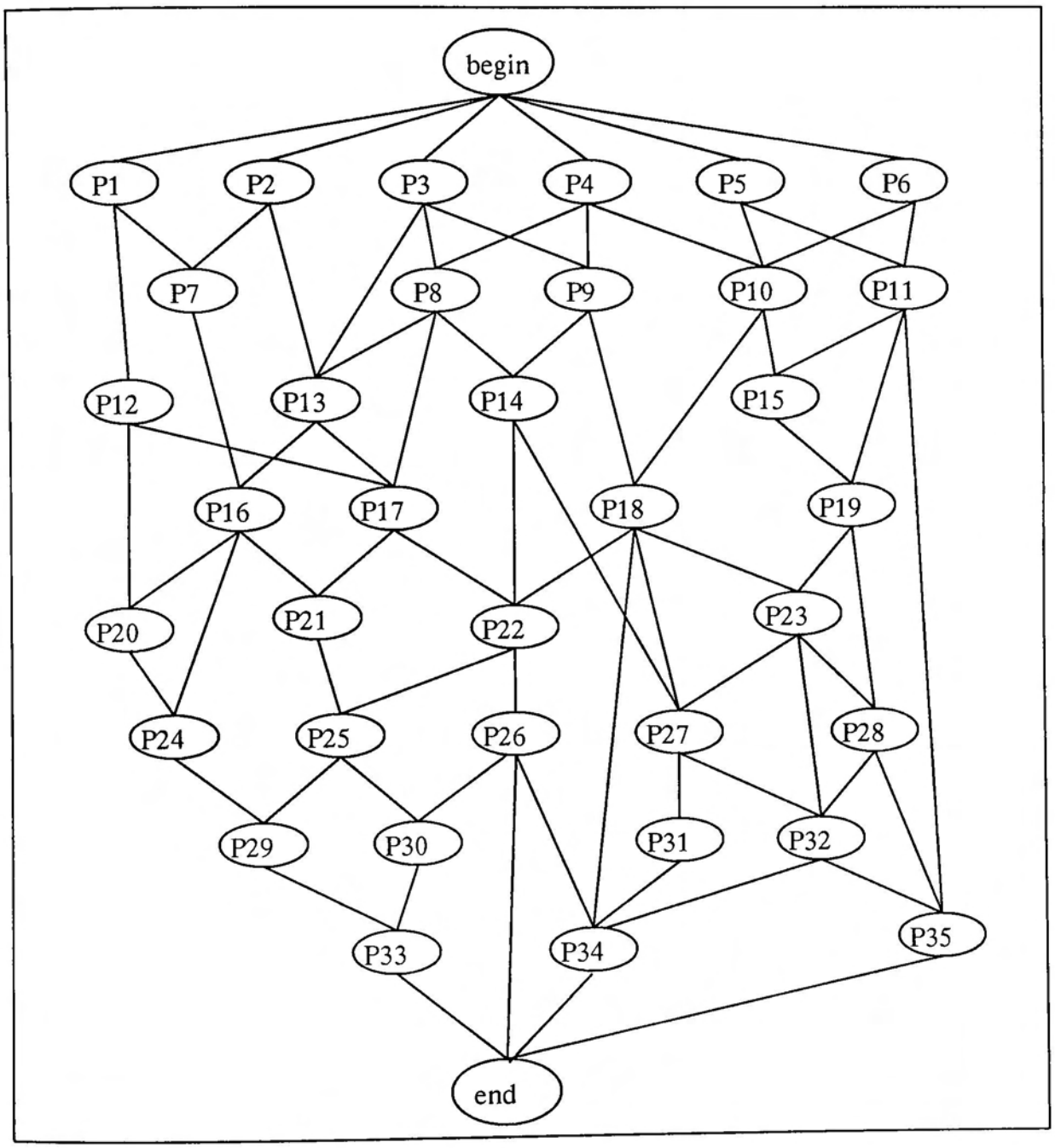

Fig.5.1. Flow graph L1 of an imaginary high-level processing.

5.2.2 Segmentation of the Image Processing Flow Graph

At present, the prototype HIGIPS consists of a PIU, three PPU's, and a POU. And each PPU is composed of a sub-controller and two slave processors, but the number of salve processors can be easily extended to eight. So from now on, we describe the software scheme under the supposition that the HIGIPS has three processing stages and each stage include eight slave processors.

To execute the above flow graph on HIGIPS, it is necessary to segment it into three parts and assign them to the processing stage 1, 2, and 3, sequentially. There are two criteria for segmentation [107, 108]. Let (x,y) represent the relationship between the processing unit x and processing unit y, e.g., (P8,P14) means that the processing unit P8 must be executed before P14. And let L1 be the original flow graph as shown in Fig.5.1, and L2 be the flow graph after segmented (see Fig.5.2).

[Criterion 1] $\mathbf{(x,y)\in L1 \Rightarrow (x,y)\in L2}$

That is, if (x,y) exists in L1, it must also exist in L2, but they can be assigned to the different part (or stage), e.g., (P14,P22) and (P11,P35) still exist even though P22 is assigned to stage 2 and P35 is assigned to stage3.

[Criterion 2] **Make** $\tau=\max\{\tau_1, \tau_2, \tau_3\}$ **be minimum.**

Where τ_1, τ_2, and τ_3 are the processing time of stage1, 2, and 3, respectively. If $\tau_1=\tau_2=\tau_3$, the segmentation is optimal.

In this case, the task assignments are as follows.

Stage 1:

Q1s, Q1e:	to the sub-controller of stage 1;
P1-P15:	to slave processors in stage 1.

Stage 2:

Q2s, Q2e:	to the sub-controller of stage 2;
P16-P28:	to slave processors in stage 2.

Stage 3:

Q3s, Q3e:	to the sub-controller of stage 3;
P29-P35:	to the slave processors in stage 3.

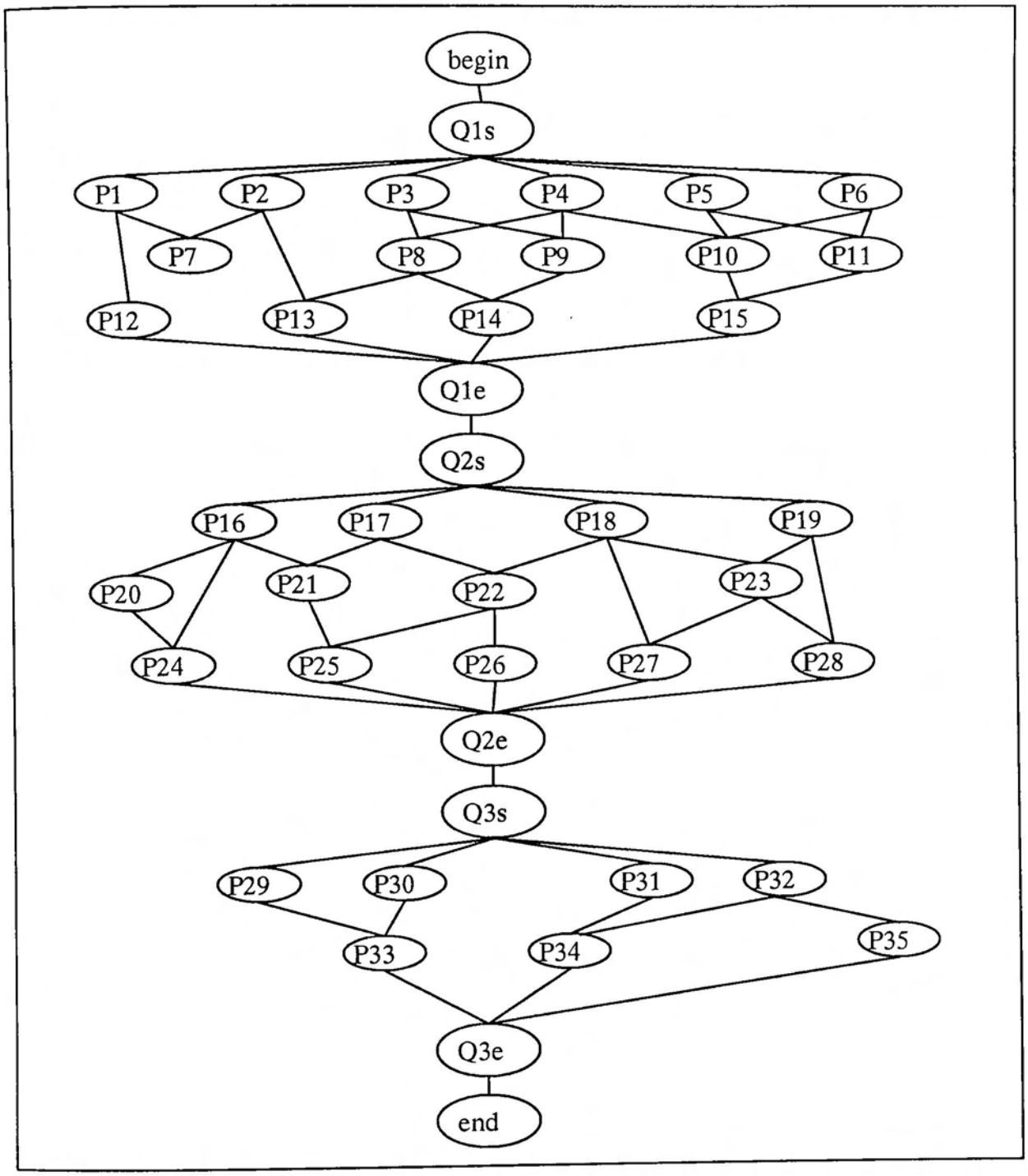

Fig.5.2. Segmented flow graph L2 of L1.

5.2.3 Stage Scheduling

As early mentioned, each stage of HIGIPS is a multiprocessor. The SMPU's operate according to the messages from SUBC. We take the stage 1 as the example to explain

how this multiprocessor works. The sub-flow graph for stage1 is redrawn in Fig.5.3, and a task relation table is obtained from it (see Table 5.1). The SUBC schedules according this table. The processing flow chart of SUBC is shown in Fig.5.4. To make the program for SUBC, the following variables and subroutines (or functions) are defined.

(1) *task_state[15]:* memorize the task status assigned to this stage;
1: waiting to be executed;
2: able to be executed;
3: being executed;
0: finished.

(2) *processor_state[8]:* memorize the processor status in this stage;
1: not using;
0: in use.

(3) *order_table[16][15]:* memorize the task relationship in Table 5.1;

(4) *clear_task_state():* set all tasks to waiting state;

(5) *get_stage_start():* receives the stage start message from SC;
0: unable to start;
1: able to start.

(6) *get_next_task():* obtain the task number able to be executed;
large than 1: the task number able to be executed;
0: there are tasks being in the waiting state but unable to be executed;
-1: all tasks are finished.

(7) *get_available_processor():* get the slave processor number able to be used;

(8) *start_task(N1,N2):* order the slave processor *N1* to execute the task *N2*, i.e., to set *task_state[N1-1]* to 3, and *processor_ state[N2-1]* to 0.

(9) *put_stage_completion():* send the "STAGE END" message to SC.

Using these variables and subroutines given above, the program for SUBC can be written as follows.

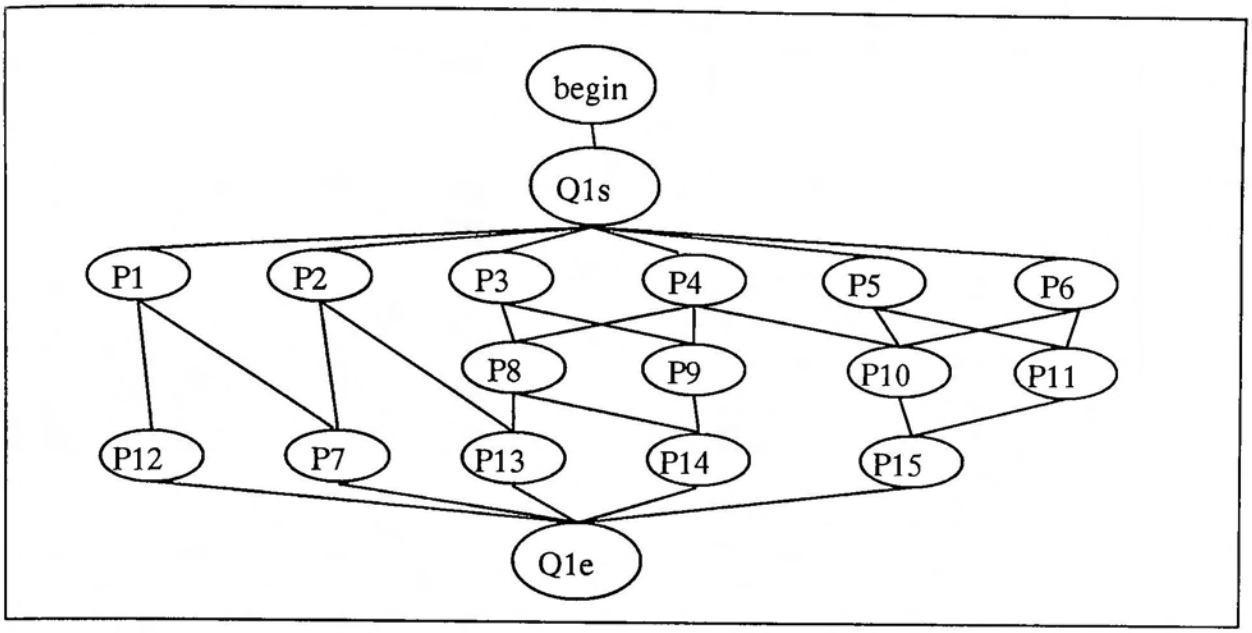

Fig.5.3. Sub-flow graph for stage 1.

Table 5.1

Task Relation Table of Sub-Flow Graph Assigned to Stage 1

↓ J / → I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1	1															
2	1															
3	1															
4	1															
5	1															
6	1															
7		1	1													
8				1	1											
9				1	1											
10					1	1	1									
11						1	1									
12		1														
13			1	1					1							
14									1	1						
15											1	1				
Q1e								1					1	1	1	1

Note: "1" means there is a direct relation between tasks, and blank means there is no direct relation between tasks.

```
char    task_state[15], processor_No, order_table[16][15];
int     processor_state[8], next_task_No;
main()
{
        clear_task_state();
        clear_processor_state();
        while (get_stage_start()==0) {
        }
        while ((next_task_No=get_next_task()) >= 0) {
                if (next_task_No > 0) {
                while ((processor_No=get_available_processor())==0) {
                }
                start_task(next_task_No, processor_No);
                }
        }
}
```

To make the program for the slave processor, following variables and subroutines are prepared.

(1) *task_No:* task number that will be executed;
"0" means no task to be executed.
(2) *get_task_No():* get the task number that will be executed from SUB;
(3) *put_task_completion():* send "END" message and processor number to SUBC;
(4) *P1()-P15():* processing subroutines. Each is a processing unit for sub-flow graph in Fig.5.3.

With these variable and subroutines, the program for slave processors can be written as follows.

```
main()
{
char    task_No;
while ((task_No=get_task_No()==0) {
}
switch (case task_No) {
        case 1: P1(); break;
        case 2: P2(); break;
```

```
        case 3: P3(); break;
        .
        .
        .
        case 14: P14(); break;
        case 15: P15(); break;
        break;
}
put_task_completion(processor_No, task_No);
task_No=0;
P1()
{
...
}
.
.
.
P15()
{
...
}
}
```

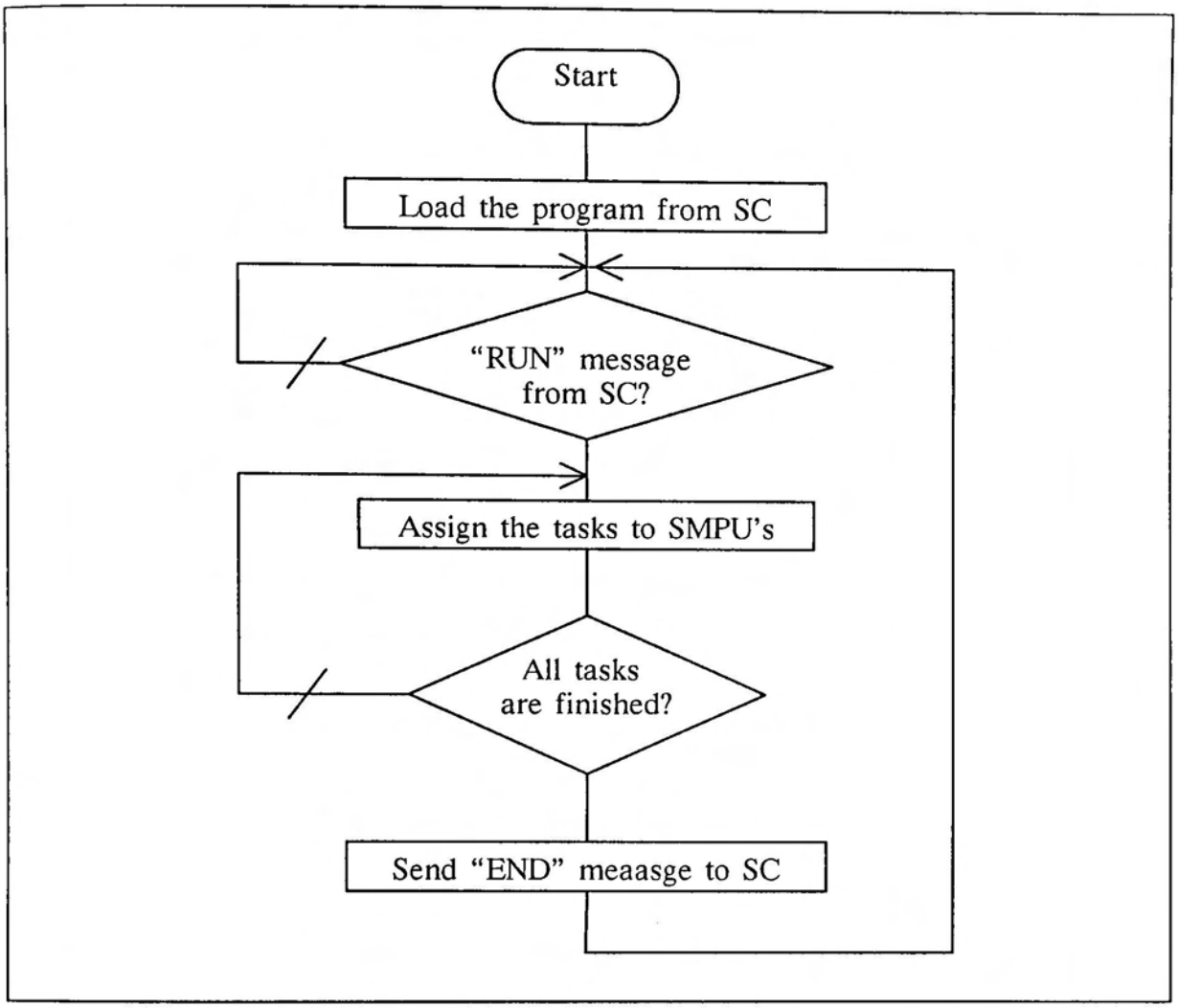

Fig.5.4. Processing flow chart of SUBC in each stage.

5.3 Communications between SUBC and SMPU in a Stage

As described above, SMPU's in a stage firstly get the task number from SUBC and execute the corresponding processing by means of subroutine (or function) call, and then send the task state, task number and processor number back to SUBC. This message exchanges between SUBC and SMPU's are done via the mailbox set at common memory (see Fig.5.5).

Mailboxes are divided into two groups: 1) transmit mailbox (from SUBC to SMPU's, e.g., #0→#i, i=1, 2, ..., 7); and 2) receive mailbox (from SMPU's to SUBC, e.g., #i→#0, i=1, 2, ..., 7). Whenever SUBC has messages to SMPU, it will first write the message into the corresponding transmit mailbox, and then send an interrupt signal to corresponding

SMPU. The SMPU will get this message and act according to this message whenever it is interrupted. This is the same for SMPU's.

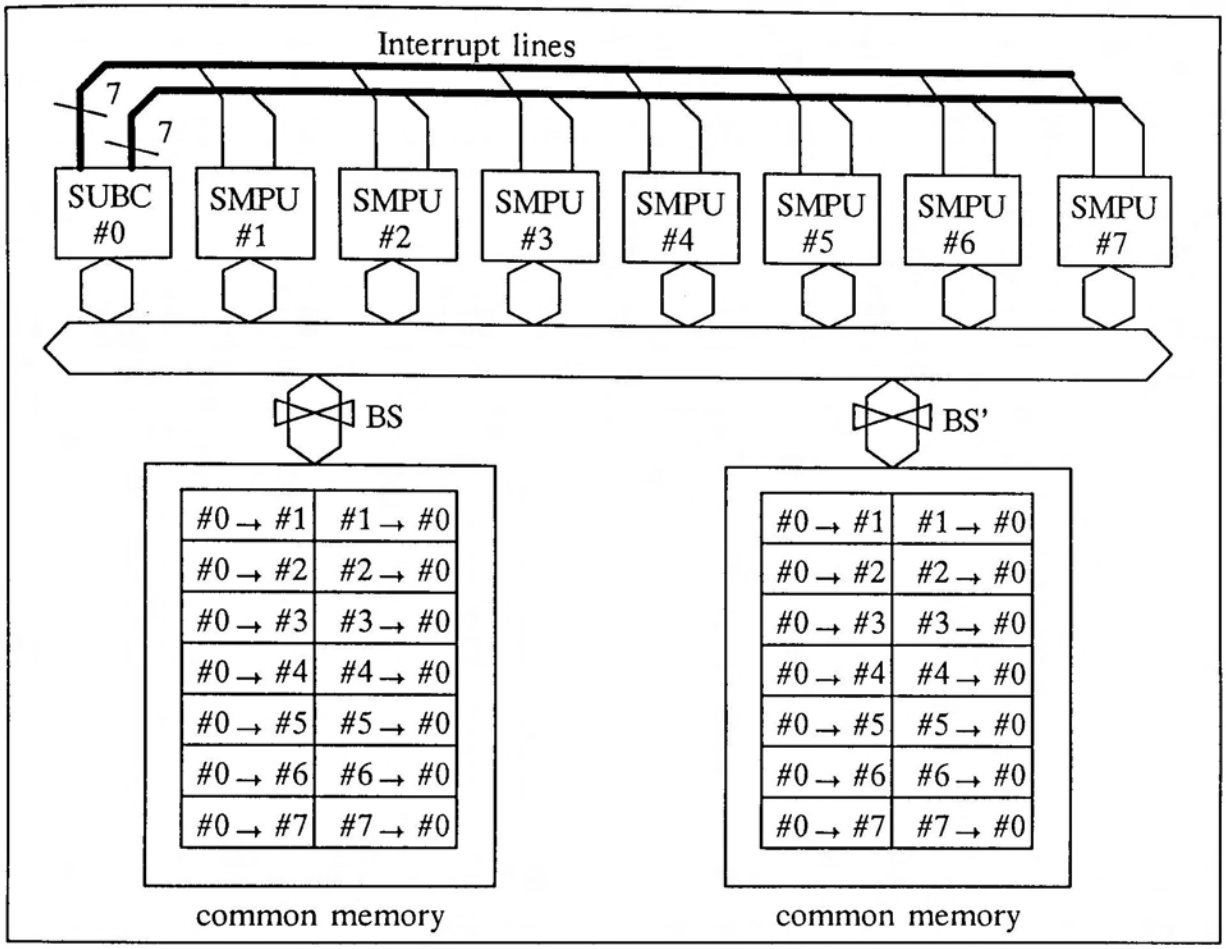

Fig.5.5. Mailboxes in global memory.

5.4 Monitor Program

To perform the processing as above, two monitor programs are needed: 1) the monitor between SC and SUBC; 2) the monitor between SUBC and SMPU.

5.4.1 Monitor Facilities between SC and SUBC

The message exchanges between SC and SUBC are done via the parallel bus (GP-IB bus). This monitor facilities are obtained by the following commands and library functions [108].

Commands in SC side:

(1) Program download

Send the processing file (EXE file) to SUB and SMPU of each stage.

(2) Run

Order each stage to start the program downloaded at the local memory.

(3) Stop the execution

Stop the program even it is being executed.

Library functions in SC side:

(1) Message exchange

Exchange the messages with SUBC in each stage to get the stage status.

(2) Polling

Monitor the stage status.

(3) Change over bus switches

When "END" message is returned back from every stage, change bus switches to another phase, and start processing on the new data.

Library functions in SUBC side:

(1) Program load

Load the processing program from SC, the program for SUBC into the local memory of SUBC, the program for SMPU's into the common memory and then transfer into the local memories of SMPU's.

(2) Execute the program

Execute the tasks received from SC.

(3) Stop

Stop processing according to stop message from SC.

(4) Exchange messages

Exchange the message between SC and SUBC, SUBC and SMPU's.

5.4.2 Monitor Facilities between SUBC and SMPU

The message exchanges between SUBC and SMPU's are done via the common memory. Let's use the segmented flow graph L2 in section 5.2 to explain how this monitor works.

SUBC side:

SUBC of stage 0 will start the task Q1s after finishing the initialization of all SMPU's in this stage. When receiving the "RUN" message, it assigns the task P1 to SMPU 1, P2 to SMPU 2, ..., P7 to SMPU 7 and starts the scheduling. So do SUBC's in stage 2 and 3. As soon as it enters Q1e, it sends "END" message to SC and apply for the shifting of image data.

SUBC of each stage schedules its tasks assigned by SC and manages its SMPU's as follows.

(1) As shown in Fig.5.6, the scheduler of user program preserves the working areas TSK_S (abbreviation for task_status), PCR_N (processor_No.) and TSK_N (task_NO.). The scheduler keeps checking these working areas.

(2) When it is interrupted by a SMPU (which finished the task and entered the waiting state), it leaves the scheduler and jumps to the interrupt processing routine. The interrupt processing routine gets the task number, e.g., *i*, (that was executed), its status, and the slave processor number.

(3) The interrupt processing routine writes them into the working areas TSK_N, TSK_S and PCR_N, and return to scheduler.

(4) The scheduler examines whether the new task, e.g., *j*, can be executed or not according these information. If it is able to be executed, to assign task *j* to processor *k*, it calls monitor, and passes *j* and *k* to monitor.

(5) The monitor program writes the task number *j* and processor *k* to the mailbox, and send a interrupt signal to the slave processor *k*, and return back to scheduler of the user program.

SMPU side:

SMPU receives the tasks or exchanges messages as follows [107].

(1) The user program of each SMPU preserves the working area TSK_N (see Fig.5.6), and the scheduler of the user program keeps checking this working area.

(2) As soon as an interrupt signal enters, it leaves the scheduler and jumps to the interrupt processing routine, The interrupt processing routine gets the task number from the mailbox, and write into the working area TSK_N, and then return to the scheduler.

(3) The scheduler of the user program gets the task number from the working area TSK_N and call subroutine P_j (task j) to perform the processing. If it is finished normally, set TSK_S to "0", and "1" in the other case.

(4) To send the task status and processor number to SUBC, the Scheduler calls the monitor and passes them to the monitor program.

(5) The monitor writes the task number and processor number into the corresponding mailbox, and send an interrupt signal to SUBC, and then return back to the scheduler.

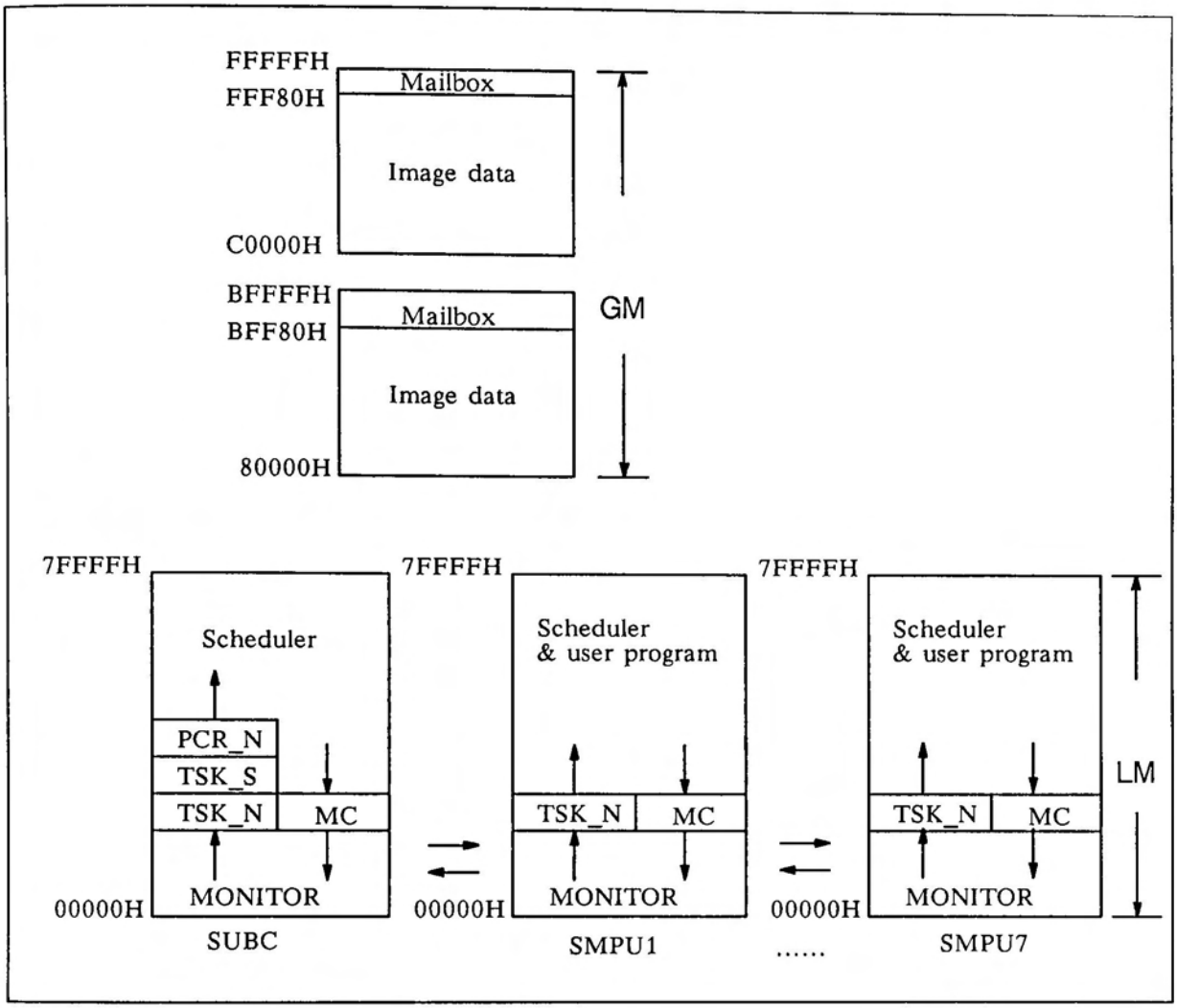

Fig.5.6. Processing control in a stage.

In next chapter, the implementation of this monitor will be discussed. And on this monitor, image processing experiments will be performed. The experimental results will be shown.

Chapter 6

EXPERIMENTS AND EVALUATION OF PROTOTYPE HIGIPS

At present, the prototype HIGIPS is working on the mini-monitor program which is partial construction of the monitor discussed in section 5.4. On this mini-monitor, image processing experiments are done, and the performance of prototype HIGIPS are evaluated. Specifications of the prototype HIGIPS are displayed in table 6.1 and some hardware characteristics of prototype HIGIPS are summarized in table 6.2. Certainly, a higher performance can be obtained if the number of processors in a stage is increased.

Table 6.1
Prototype HIGIPS Specifications

Items	Standards
Image data	Static image or dynamic image scenes
Processor	General-purpose processor μPD70216(V50) for PIC, POC, SUBC's and SMPU's
CPU clock	8-MHz
Bus clock	10-MHz
Possible devices to connect	TV camera, VTR and analog display

Table 6.2
Some Hardware Characteristics of Prototype HIGIPS

Characteristics	Individual Processor	Prototype HIGIPS ($N=3, M=2$)
Instruction/s (maximum)	4.0×10^6	3.6×10^7
Multiplies or divides/s	2.0×10^5-2.5×10^5	1.8×10^6-2.25×10^6
Pipeline bus rate (bytes/s)	8.42×10^5	8.42×10^5
Program load rate (bytes/s)	1.0×10^6	1.0×10^6

6.1 GP-IB Interface Library Fuctions

The system controller (SC) of the prototype HIGIPS controls SUBC's in every stage via the GP-IB parallel bus. And it controls all SMPU's indirectly via SUBC. The kernel of the monitor program is the construction of these GP-IB interface library functions. Thereby, before explaining the construction of the monitor programs on SC and SUBC (and SMPU's), let's firstly explain how to construct the GP-IB interface library functions. All these functions are listed in table 6.3.

Table 6.3
GP-IB Interface Library Functions

No.	Name of Functions	Descriptions
1	*gpib_int*	Initialize μPD7210
2	*gpib_ifc*	Send IFC signal
3	*tm100*	Timer with unit of μ100
4	*gpib_ren*	Set REN signal
5	*gpib_clrren*	Clear REN signal
6	*gpib_delim*	Definition of delimeter
7	*gpib_rdst*	Read address status register
8	*gpib_cmd*	Send commands
9	*gpib_write*	Output data block
10	*gpib_wrt*	Output one byte datum
11	*gpib_read*	Input data
12	*gpib_spoll*	Serial poll execution
13	*gpib_spst*	Read serial poll register
14	*gpib_srq*	Service request
15	*gpib_ppol*	Assignment of parallel poll response lines
16	*gpib_idy*	Parallel poll execution
17	*gpib_ppr*	Assignment of parallel poll response mode
18	*gpib_pp2*	Assignment of parallel poll response line at pp2

These functions can be shortly described as follows:

(1) Initialization of μPD7210

Operations: To initialize μPD7210 and set the initial value for the external variables those are used in GP-IB.

Function Name: *gpib_int*

Format: *void gpib_int()*;

(2) System Initialization

Operations: After initialization, μPD7210 enters the slave mode state but it does not have the functions of controller. To make it act as a controller, it is necessary to set IFC line *on* for at least more than 100-μS. This function sets IFC line *on* for about $n \times 100$-μS.

Function Name: *gpib_ifc*

Format: *int gpib_ifc(n)*;
int n;

Return Value: If it is a master, IFC is sent out and 0 is returned. Otherwise, doing nothing except 1 is return.

(3) 100-μS Timer

Operation: For the parameter *n*, it return after $n \times 100$-μS is past.

Function Name: *tm100*

Format: *void tm100(n)*;
int n;

(4) Control of REN Signal

(a) Set REN Signal *ON*

Operation: It is only necessary for master GP-IB device to set REN *ON/OFF*. Setting REN *ON* will make other GP-IB devices enter remote state.

Function Name: *gpib_ren*

Format: *int gpib_ren()*;

Return Value: For the master GP-IB device, REN is set *ON* and 0 is returned. For the slave GP-IB devices, 1 is returned and doing nothing.

(b) Set REN *OFF*

Operation: Setting master GP-IB device REN *OFF* will make other GP-IB device enter the local state.

Function Name: *gpib_clrren*

Format: *int gpib_clrren()*;

Return Value: For the master GP-IB device, REN is set *OFF* and 0 is returned. For the slave GP-IB devices, 1 is returned and doing nothing.

(5) Delimiter Definition

Operation: To define the delimiter code when doing data block transferring. Four kinds delimiters are shown by the integer *n* as follows:
n=0: CR+LF code;
n=1: CR code;
n=2: LF code
n=3: without delimiter.

Function Name: *gpib_delim*

Format: *void gpib_delim(n);*
int n;

(6) Reading of Address Status Register

Operation: To read the address status register. Each bit of it has the following meanings.

mjmn	: It becomes to 1 when receiving minor address;
ta	: It becomes to 1 when GP-IB device acts as a listener;
la	: It becomes to 1 When GP-IB device acts as a talker;
tpas, lpas	: They are the flags of address mode 3:
spms	: It becomes to 1 when serial poll is enabled;
iatn	: It becomes to 1 when ATN line is high;
cic	: It becomes to 1 when GP-IB device is set to the controller.

Function Name: *gpib_rdst*

Format: *void gpib_rdst();*

Return Value: Value of the address status register.

(7) Sending out Commands

Operation: To send out *n* commands preserved in character array *buf.*

Function Name: *gpib_cmd*

Format: *void gpib_cmd(n, buf);*
int n;
*unsigned char * buf;*

Return Value: For the master GP-IB device, 0 is returned after commands are sent out. For slave GP-IB devices, 1 is returned and doing nothing.

(8) Output Data

Operation: To send out *n*-byte data preserved in character array *buf.* After *n*-byte data are sent out, the delimiter (if defined, i.e., 0, 1, and

2) is also sent out. If *eoi* is 1, EOI line is also *ON* at the end of data sending.

Function Name: *gpib_write*

Format: *void gpib_write(n, buf, eoi);*
int n, eoi;
*unsigned char * buf;*

Return Value: 0 is returned when data sending is finished correctly. Otherwise, 1 is returned.

(9) Output One Byte Datum

Operation: To send out *1*-byte datum *c*.

Function Name: *gpib_wrt*

Format: *void gpib_wrt(c);*
char c;

Return Value: 0 is returned when *1*-byte datum sending is finished correctly. Otherwise, 1 is returned.

(10) Input Data

Operation: To read data from GP-IB interface and preserve into character array *buf*. This will continue until *n*-byte data are received, or delimiter or *EOI* message are received.

Function Name: *gpib_read*

Format: *int gpib_read(n, buf);*
int n;
*unsigned char *buf;*

Return Value: Number of data bytes preserved in *buf*.

(11) Execution of Serial Poll

Operation: When *srqi*-bit of interrupt status register 1 becomes 1, the serial poll is started for the GP-IB devices whose address number are preserved in *buf*. Whenever the device that requests service is detected, the address of this device is preserved in *dev*, and device status is preserved in *sps*. Only the master GP-IB device can execute the serial poll.

Function Name: *gpib_spoll*

Format: *int gpib_spoll(n, buf, dev, sps);*
*int n, *dev, *sps;*
*unsigned char *buf;*

Return Value: 0 is returned when serial poll is executed;

1 is returned if GP-IB device works in slave mode and cannot do serial poll.

2 is returned when *srqi*-bit is 0 and without doing serial poll.

(12) Reading of Serial Poll Status

Operation: To read the serial poll status register. This is for verifying whether the controller device receives the slave device status or not when the slave device requests service. That is, to check *pend*-it, if it is 1, the controller did not receive the device status; if it is 0, the controller already received the device status.

Function Name: *gpib_spst*

Format: *int gpib_spst();*

Return Value: Value of serial poll status register.

(13) Service Request

Operation: The slave device requests the service to controller. Preserve the device status in *n*, set *rsv*-bit of serial poll mode register to 1, and move content of *n* to *spstatus*. If *eoi* is 1, EOI line is set *ON* when sending device status.

Function Name: *gpib_srq*

Format: *int gpib_spst(eoi, n);*
int eoi, n;

Return Value: 0 is returned when the slave device requests the service and done. 1 is returned and nothing is done when the device is in the master mode.

(14) Assignments of Parallel Poll Response Lines

Operation: For the devices which have pp1 subset of parallel poll functions, the controller assigns the parallel response line to them. Preserve |n| device number and their response line number in *buf* and *bufd*. If *n* is negative, PPU command is sent out before the assignments of response lines, and this releases the present assignments of parallel poll response lines. Data preserved in *bufd* is called PPE or PPD command.

Function Name: *gpib_ppol*

Format: *int gpib_ppol(n, buf, bufd);*
int n;
*unsigned char *buf, *bufd;*

Return Value: When the device works in the master mode, the assignments of parallel poll response lines are done, and 0 is returned. If the device works in slave mode, 0 is returned and nothing is done.

(15) Execution of Parallel Poll

Operation: To execute the parallel poll and obtain its result.
Function Name: *gpib_idy*
Format: *int gpib_idy()*;
Return Value: Parallel poll result 0-255 for the master device, -1 for slave device.

(16) Response to Parallel Poll

Operation: To assign the parallel response mode according the value of *n*.
n=0: Do not response the parallel poll;
n=1: Response the parallel poll;
n=2: Response parallel poll whenever there is a service request.
Function Name: *gpib_ppr*
Format: *int gpib_ppr(n)*;
int n;
Return Value: When the response for parallel poll is done, 0 is returned. 1 is returned for the master device.

(17) Assignments of Parallel Poll Response Lines for Device with PP2 Function

Operation: To assign the parallel poll response lines for the device which has pp2 subset of parallel poll function. The assignments of parallel poll response lines and response mode are done according to *u*, *s*, *p*.
u=0: Response parallel poll by the value of *s* with *p+1* line;
u=1: Do not response parallel poll.
Function Name: *gpib_pp2*
Format: *int gpib_pp2(u, s, p)*;
int u, s, p;
Return Value: When the response for parallel poll is done, 0 is returned. 1 is returned for the master device.

With these basic GP-IB interface functions, data input/output via GP-IB can be realized easily.

6.2 Terminal Program on SC

Terminal program on SC is constructed with use of above basic library function Here, let's introduce some example combinations of them, and explain the constructi of this terminal program.

6.2.1 Data Input/Output Library Functions via GP-IB Interface

(1) Initialization

Operation: To initialize μPD7210 and GP-IB system. This initialization is to set IFC *ON* for a period of time, and then set REN *ON*. This is simply to run *gpib_int*, *gpib_ifc*, and *gpib_ren*, sequentially. This must be done before using GP-IB. And it can be defined as a new function in the following.

Function Name: *gopen*

Format: *int gopen();*

Return Value: 0 is returned for master device, and 1 is returned for slave devices.

Program:

```
gopen()
{
        gpib_int();
        gpib_ifc();
        return(gpib_ren());
}
```

(2) Data Output

Operation: To send out *n*-byte data preserved in *buf* to the device numbered as *lsna*. This is simply combination of *gpib_cmd* and *gpib_write*.

Function Name: *gwrite*

Format:

```
int gwrite(lsna, n, buf, eoi);
int lsna;   /* Lintener address number */
int n;      /* Data length */
unsigned char *buf; /*Data buffer */
int eoi;    /* EOI parameter */
```

Return Value: 0 is returned if it is finished correctly, otherwise, 1 is returned.

Program:

```
gwrite(lsna, n, buf, eoi)
int lsna, n, eoi;
unsigned char buf[];
{
```

```
        char cmd[3];
        cmd[0] = 0x3f;
        cmd[1] = mytka;
        cmd[2] = 0x20 + lsna;
        gpib_cmd(3,cmd);
        return(gpib_write(n, buf, eoi));
}
```

A function to send out character string data can be defined as follows:

Function Name: *gswrite*

Format:
```
int gswrite(lsna, buf, eoi);
int lsna;   /* Lintener address number */
unsigned char *buf; /*Data buffer */
int eoi;    /* EOI parameter */
```

Return Value: 0 is returned if finished correctly, otherwise, 1 is returned.

Program:
```
gswrite(lsna, buf, eoi)
int lsna, eoi;
unsigned char buf[];
{
        return(gwrite(lsna, strlen(buf), buf, eoi));
}
```

(3) Data Input

Operation: To read *n*-byte data from GP-IB device numbered as *tka* and preserve in *buf*. This is simply combination of *gpib_cmd* and *gpib_read*.

Function Name: *gread*

Format:
```
int gread(tka,n,buf);
int tka;    /* Talker device number */
int n;      /* Data length read from GP-IB interface */
unsigned char *buf; /* Data buffer */
```

Return Value: Data length read from GP-IB interface.

Program:
```
gread(tka, n, buf)
int tka, n;
unsigned char buf[];
{
        char cmd[3];
        cmd[0] = 0x3f;
        cmd[1] = 0x40 + tka;
```

```
        cmd[2] = mylsna;
        gpib_cmd(3, cmd);
        return(gpib_read(n, buf));
}
```

A function to receive character string data can be defined as follows:

Operation: To read data from GP-IB device numbered as *tka*, and preserve in *buf*.

Function Name: *gsread*

Format:
```
int gsread(tka, buf);
int tka;    /* Talker device number */
unsigned char *buf; /* Data buffer */
```

Return Value: Data length read from GP-IB interface.

Program:
```
gsread(tka, buf)
int tka;
unsigned char buf[];
{
        int i;
        i = gread(tka, 1024,buf);
        if (delim == 0) i = i-2;
        if (delim ==1 || delim == 2) i = i-1;
        buf[i] = 0;
        return(i);
}
```

6.2.2 Terminal Program Commands

There are 9 kinds of commands are constructed for the terminal program. They are described in the following.

(1) Memory Write Command

This command is to write data from SC into the local memory of SUBC or global memory (common memory) in every stage. Its format is shown in Fig.6.1. Note here, from now on we use hexadecimal to represent numbers instead of decimal.

The *1*-st byte is called command byte. The upper 4-bit (b7-b4) of command byte is to represent the command type (numbered from **0** to **f**), the lower 3-bit (b2-b0) is to represent the slave processors (numbered from **1** to 7), and b3 of it is always set to 1 to avoid the command byte to **00**. For the memory write command, the command byte is **08** (hexadecimal).

The 2-nd to 5-th bytes are the *segment* and *offset*. Each takes two bytes. Because the processors of HIGIPS are μPD70216 which has 1-megabyte memory space, 20-bit address lines are necessary. These 20-bit address lines are divided into *segment* and *offset*. The terminal program on SC makes this *segment* and *offset* from a input 20-bit address. For example, when the input address is *3f4ea*, the terminal program will separate it into *3000* and *f4ea* as *segment* and *offset* respectively.

The 6-th and 7-th bytes are the data length to write. And from 8-th byte to *263*-th are used as data buffer. The size of data buffer can be maximally 256 bytes. Following the data buffer is 1 byte used as check sum. The check sum is calculated from the 2-nd byte to the end of data buffer.

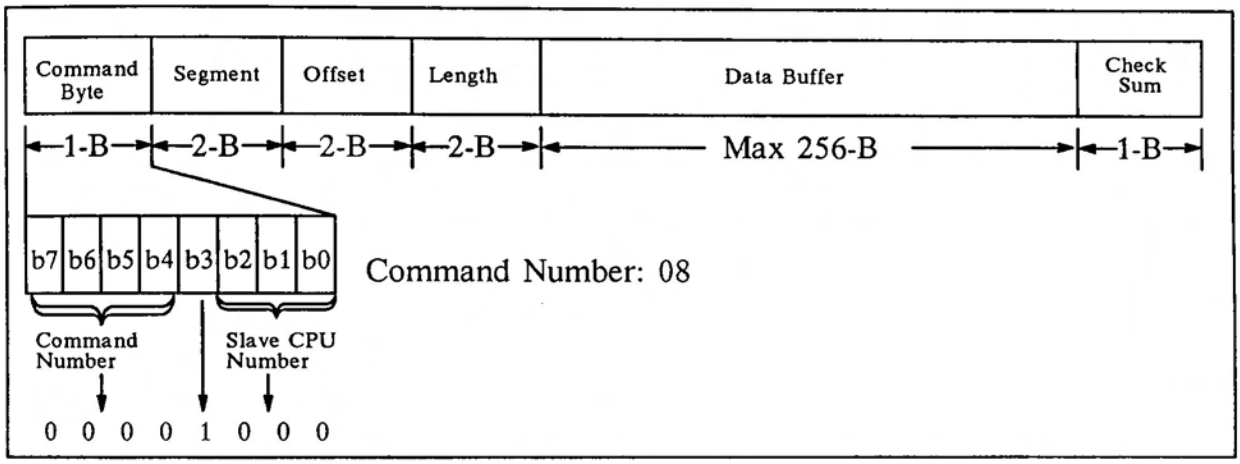

Fig.6.1. Memory write command format.

(2) Memory Read Command

This command is to read the local memory of SUBC or the global memory in every stage of HIGIPS. Its format is shown in Fig.6.2. The command number is **18**.

Nothing is different with that of memory write command except that command number and check sum byte are changed.

When SUBC in a stage of HIGIPS receives this command, it prepares the data and answers SC with memory-read-response command whose format is presented in Fig.6.3.

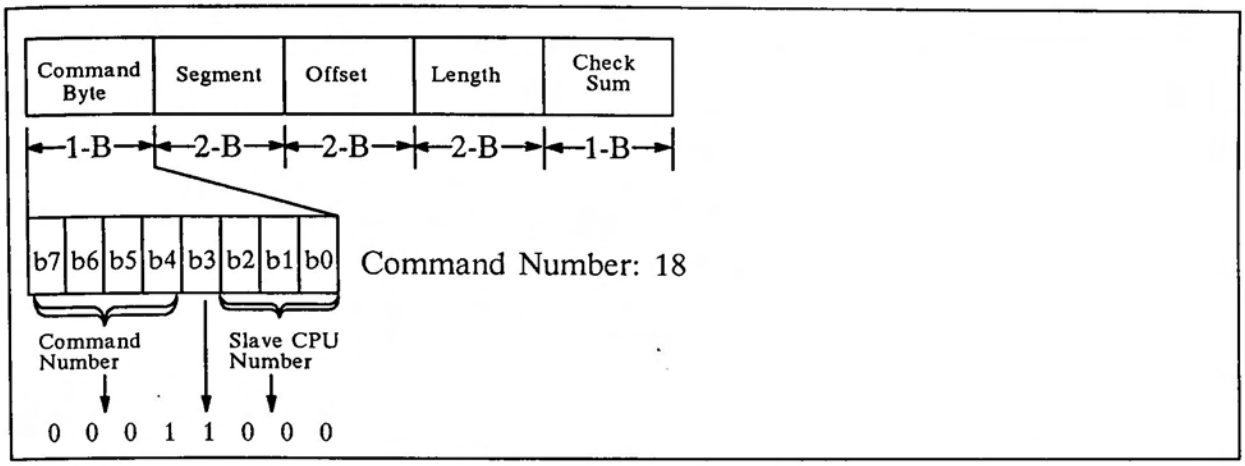

Fig.6.2. Memory read command format.

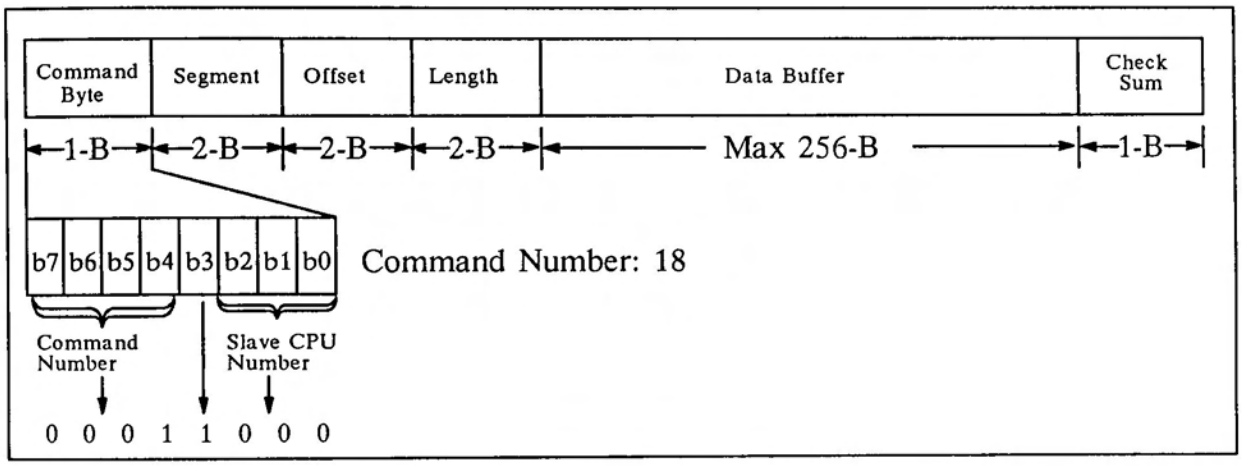

Fig.6.3. Memory-read-response command format.

(3) I/O Write Command

This command is to output data to I/O device of SUBC in every stage of HIGIPS. Its format is given in Fig.6.4. The command number is **28**.

Because I/O device of μPD70216 is assigned to 64-kilobyte address space, 16-bit address lines are enough. So only *2*-nd and *3*-rd bytes are used to indicate the I/O device address. 4-*th* and 5-*th* byte are set to **0000**. And since only 1-byte datum can be output to I/O device at a time, the data length is set to **0001**. 1-byte datum follows the data length. The final is the check sum byte.

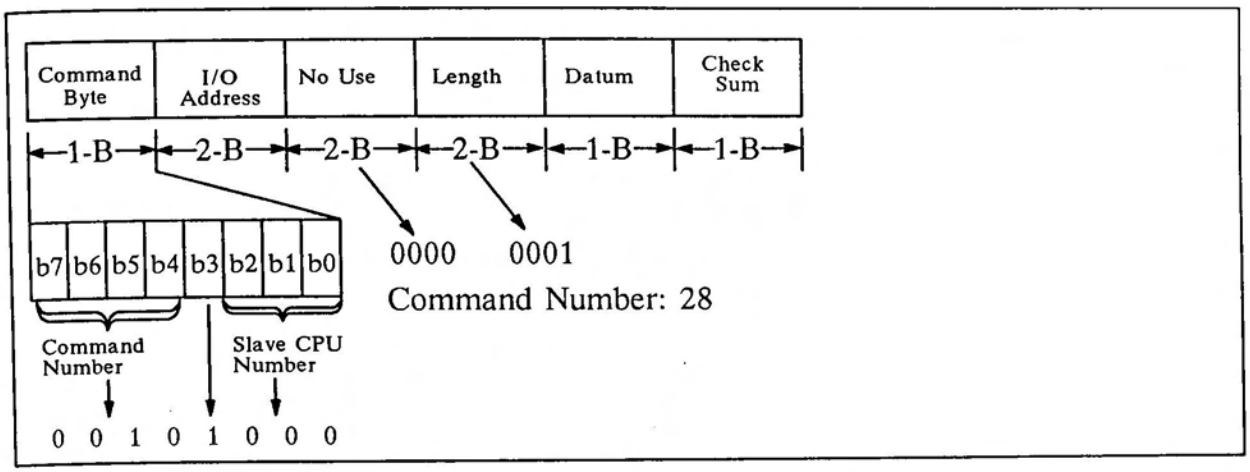

Fig.6.4. I/O write command.

(4) I/O Read Command

This command is to input data from I/O device of SUBC in every stage of HIGIS. Its format is displayed in Fig.6.5. The command number is **38**. Nothing is different with I/O write command except that it does not include one-byte datum.

When SUBC receives this command, it reads the corresponding I/O device to obtain one-byte datum. It answers SUBC with I/O-read-response command (See Fig.6.6).

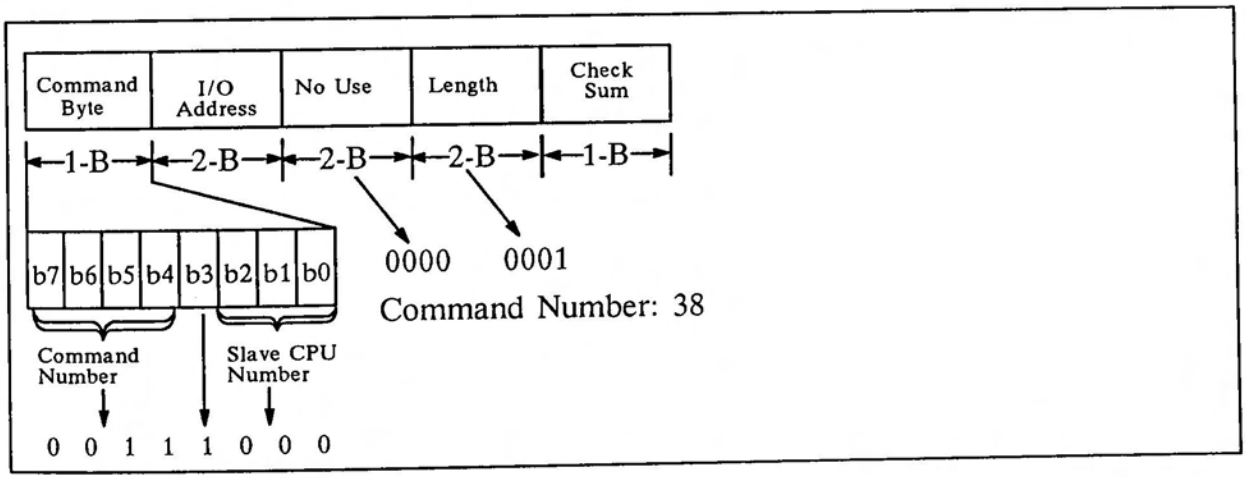

Fig.6.5. I/O read command

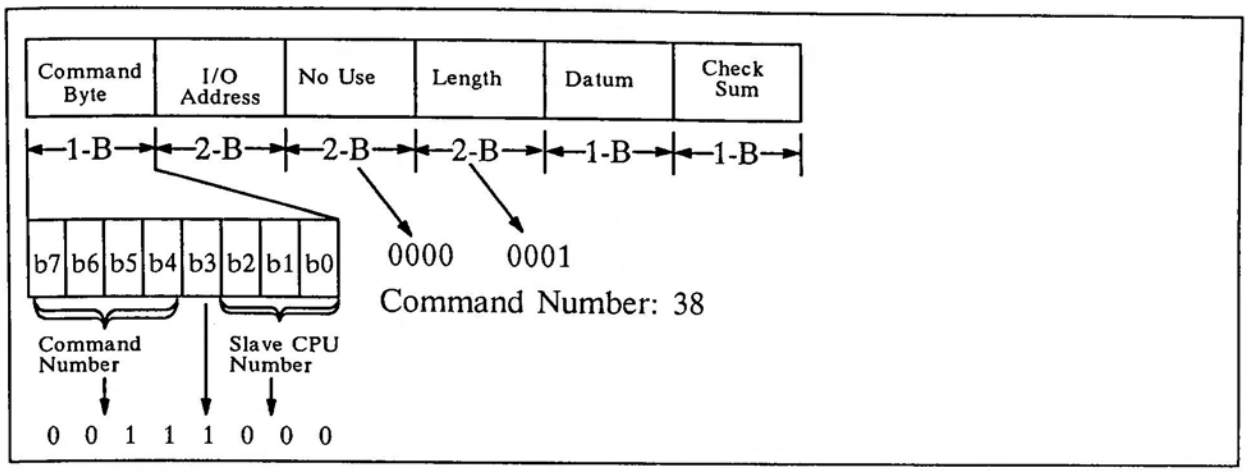

Fig.6.6. I/O-read-response command.

(5) Program Down-Load Command

This command is to load the processing program from SC to the global memory in every stage of HIGIPS. The processing program is in the form of HEX code which is made from EXE file on SC with Hextool, and can be executed directly in HIGIPS. Fig.6.7 shows the format of this command. The command number is **48**.

To down-load a HEX code file to HIGIPS is just the repeat of this command. But at each repeat, the *offset* must be increased with a value of the record length. This is continued until all records of HEX code file are down-loaded.

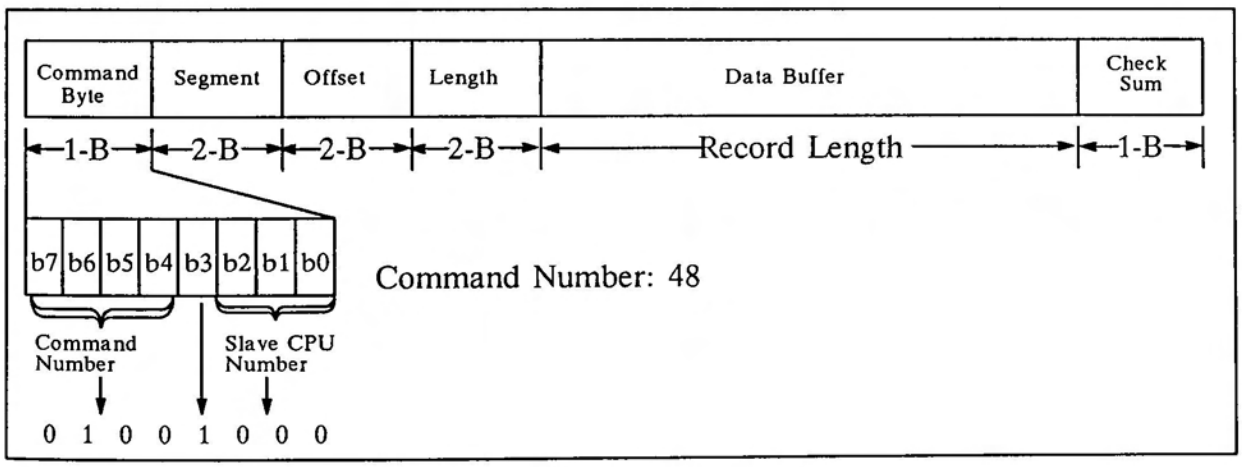

Fig.6.7. Program down-load command.

(6) Program Moving Command

This command is to order the processors in every stage of HIGIPS to move the processing program from the global memory into their local memory. Its format is given in

Fig.6.8. The *2*-nd and *3*-rd bytes are the *source segment*, and *4*-th and *5*-th bytes are the *source offset*. The *6*-th and *7*-th bytes are data bytes (here, it is **0006**). The *8*-th and *9*-th bytes are the *destination segment*, and *10*-th and *11*-th byte are the *destination offset*. The *12*-th and *13*-th are the program length. The command number is **58**.

If the down-load program is for SUBC, the slave processor bits (b2-b0) must be 0. And if the down-load program is for SMPU's, the slave processor bits must not be 0. It can be any value from 1 to 7.

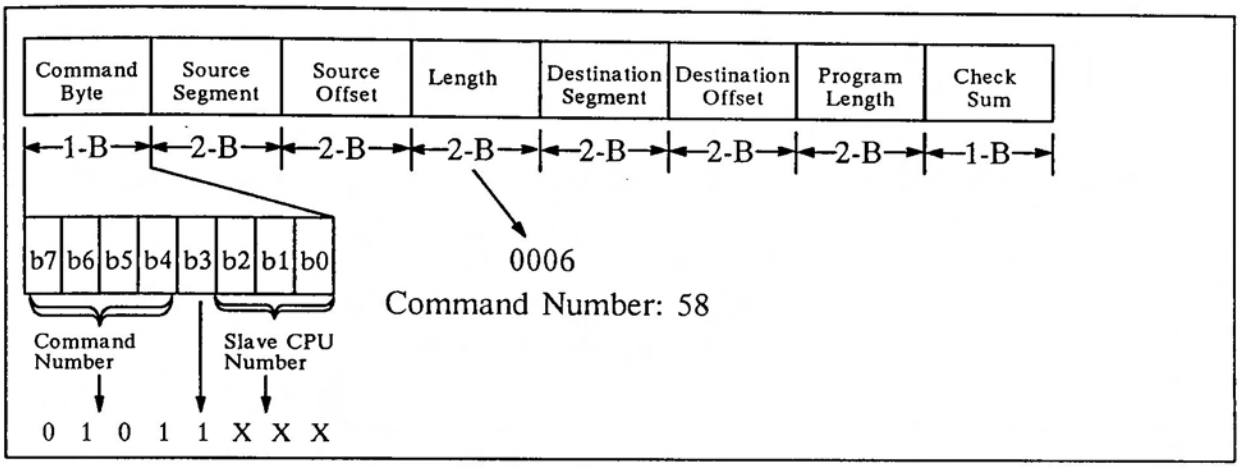

Fig.6.8. Program moving command.

(7) Program Execution Command

This is the command (See Fig.6.9) to run the down-load program (which is moved into the local memory), located from *segment:offset*. This command is only for SUBC to start to run the program in its local memory. The message for SMPU's to start to run their programs will be sent out from the program for SUBC after SUBC starts the execution.

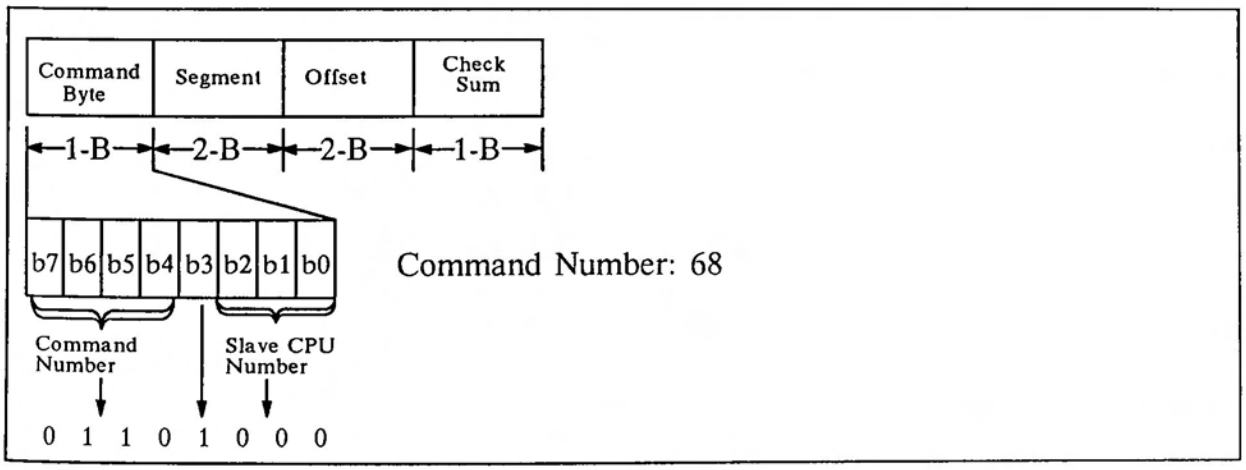

Fig.6.9. Program execution command.

(8) Start-up Testing Command

This is to check whether the corresponding stage gets ready or not (refer to Fig.6.10). If it gets ready, SC will receive a message "is ready !". Therefore, if this command is sent to PIU or POU, a combined message "PIU is ready !" or "POU is ready !" will be shown on the display of SC.

This special command comprises only on byte. And it is only for SUBC in every stage of HIGIPS, so the command byte is **00.**

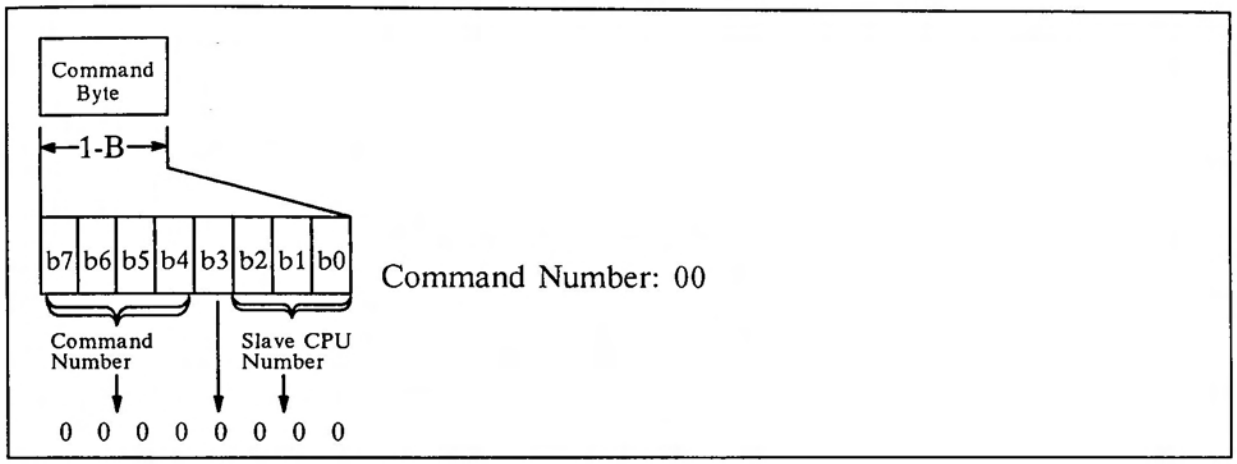

Fig.6.10. Start-up testing command.

(9) Finish Command

This command is to force all processors in a stage going back to the monitor program. As shown in Fig.6.11, it is one-byte command. The command number is f8.

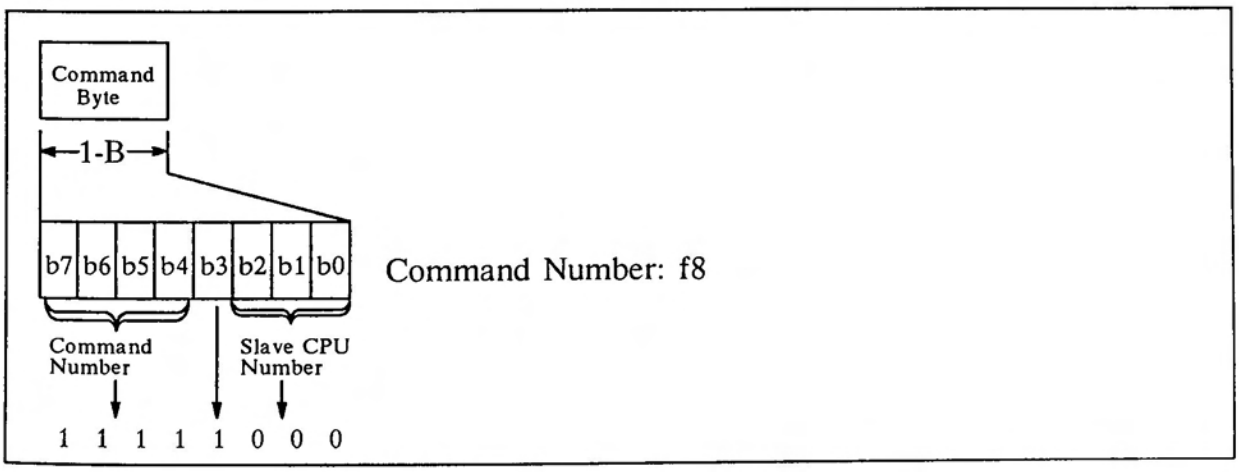

Fig.6.11. Finish command.

6.3 Mini-Monitor Program on HIGIPS

The monitor discussed in section 5.4 is partially constructed and is given a name as mini-monitor. Fig.6.12 shows its memory map.

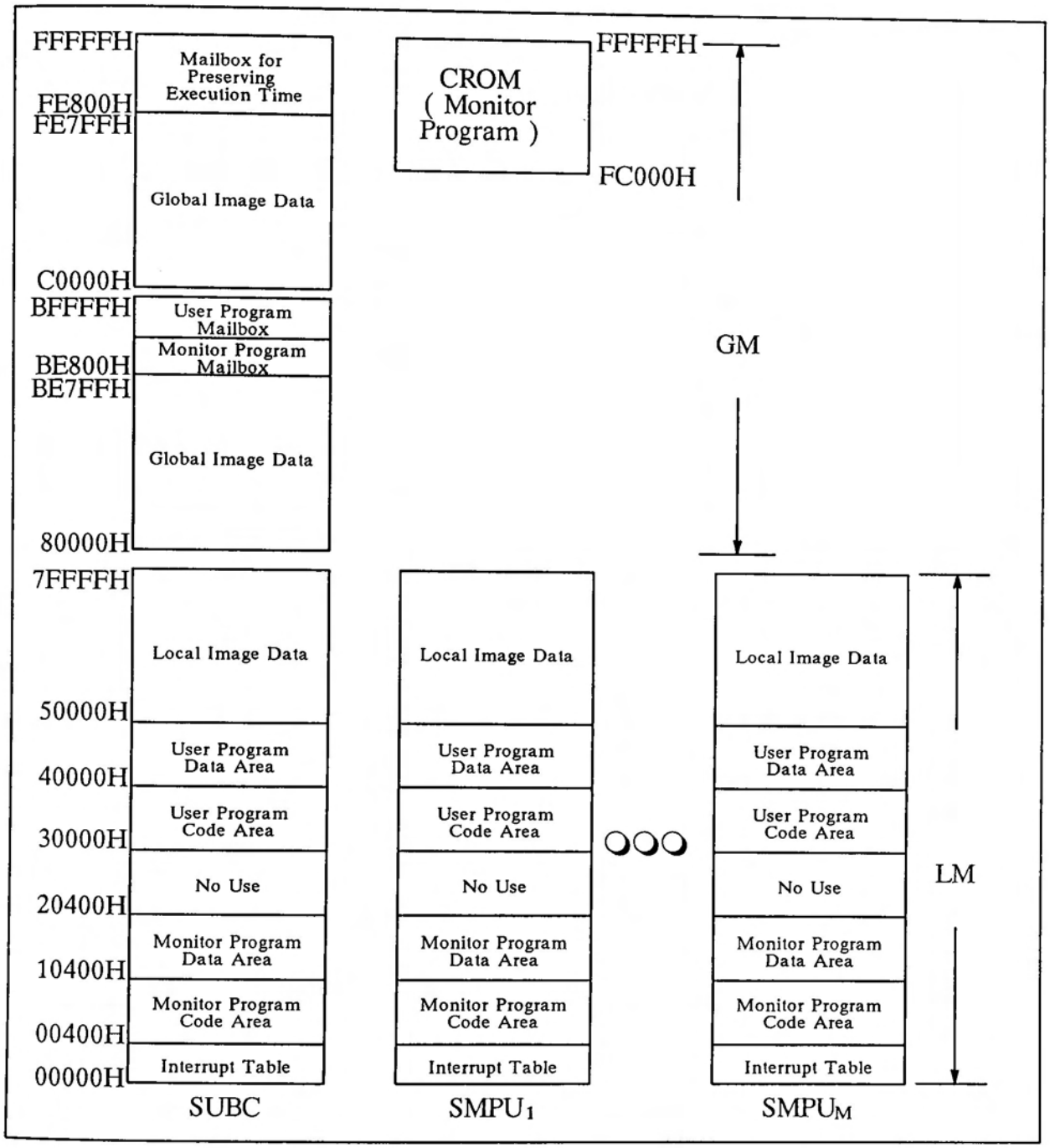

Fig.6.12. Memory map of mini-monitor program.

All processors (SUBC and SMPU's) in the same stage are using the same mini-monitor program. The most significant bit (b7) of port **A** of the parallel programmable interface (8255) is used to identify whether the processor is SUBC or SMPU. When the power is on, all processors in the same stage go to access CROM, and move the mini-monitor into their local memory, and start-up from the local memory. If a processor senses that it is set as a SMPU, it will get into a loop to receive the message from SUBC. Likewise, if a processor senses that it is set as a SUBC, it will get into a loop to receive command from SC. The mini-monitor program has the following library functions (subroutines).

(1) Memory Write Library Function

This library function is corresponding to the memory write command of SC. When the mini-monitor program of SUBC receives the memory write command, it will call this function and input the data from GP-IB interface, and place them into the stage global memory or the local memory of SUBC (refer to Fig.6.1).

(2) Memory Read Library Function

This library function corresponds to the memory read command of SC. When SC sends the memory read command (refer to Fig.6.2) to a SUBC, the mini-monitor program of SUBC will call this function. This function prepares the data, and sends them back to SC (refer to Fig.6.3).

(3) I/O Write Library Function

It is for replying the I/O write command of SC. When the mini-monitor program of SUBC receives I/O write command, it will call this function. This function outputs the received one-byte datum to the I/O device (8255) of SUBC (refer to Fig.6.4).

(4) I/O Read Library Function

This is to answer the I/O read command of SC. When the mini-monitor program of SUBC receives the I/O read command (see Fig.6.5), it will call this function. This function inputs one-byte datum from I/O device (8255) and sends it to SC (refer to Fig.6.6).

(5) Program Down-Load Library Function

This function is corresponding to the program down-load command of SC. The mini-monitor program of SUBC will call this function when it receives the program down-load command from SC. This function receives the processing program and places in the stage global memory (refer to Fig.6.7).

(6) Program Moving Library Function

This function is corresponding to the program moving command of SC. The mini-monitor program will call this function when it gets the program moving command from

SC. If the slave CPU number bits (b2-b0) of command byte is equal to 0 (refer to Fig.6.8), the received processing program is for SUBC and it will be transferred into the local memory (private memory) of SUBC. If the slave CPU number is not equal to 0, then the received processing program is for SMPU's. In this case, SUBC sends an interrupt signal to every SMPU, the mini-monitor program of SMPU will transfer the received processing program in global memory into the local memory of SMPU. All SMPU's in a stage will run the same processing program.

(7) Program Execution Library Function

This function corresponds to the program execution command of SC. The mini-monitor program of SUBC will call the program execution library function when it receives the program execution command from SC. This library function is to order SUBC to start to run. The processing program of SUBC will send a signal (an interrupt signal) to let SMPU's start to run the program (refer to Fig.6.9).

(8) Start-Up Testing Library Function

This library function is corresponding to the start-up testing command of SC. The mini-monitor program of SUBC will call this function when it receives the start-up testing command from SC. If SUBC is ready to receive the command from SC, it will send a message "is ready !" to SC (refer to Fig.6.10).

(9) Finish Library Function

This library function is corresponding to the finish command of SC. The mini-monitor program will call this function when it receives the finish command from SC. This function will force all the processors in a stage to go back to the mini-monitor program even they are running the user processing program.

So far, we related the construction of mini-monitor program for the prototype HIGIPS. From the next section, we will discuss the image processing operations on this prototype HIGIPS.

6.4 Evaluation of Prototype HIGIPS (Performance Analysis)

Each stage of the prototype HIGIPS is a common bus type multiprocessor. One weak point of this kind of system is that the bottle-neck phenomenon happens when the processors access the global memory. But in HIGIPS, this is well avoided by the software technique. The method employed here is: (1) the processors in a stage only transfer the image data blocks that they will process, into their local memories; (2) they only access the local memory when performing the assigned task; (3) they place the processed image data to the global memory of the next stage. With this method, the accessing to the global memory is necessary only when the processors transfer the image data from the global memory into their local memories and when they output the intermediate results to the global memory of the next stage. This greatly reduced the accessing frequency to the global memory. If total accessing time of the global memory is much more smaller than processing time of the tasks assigned, we can say that the bottle-neck phenomenon does not exist.

On the other hand, image processing can be classified into preprocessing, intermediate processing, and main processing, according to the processing procedure. The preprocessing is based on signal-level, and it must deal with all the pixels in an image frame. Therefore, the accessing to the global memory is most often. With respect to this, the main processing is based on the symbol-level, and it deals with only the interest pixels (e.g., the feature points). So, the accessing of the global memory is not as often as in the preprocessing. The intermediate processing performs signal-based processing or symbol-based processing in correspondence with the purpose of processing.

To perform an image processing task by a N-stage HIGIPS, it is necessary to segment this task into N sections and assign each section to a stage of HIGIPS sequentially from stage 1 to stage N. Each stage performs the same operation to the image data frame or the intermediate result passed from the previous stage, according to the command from SC. An image frame is processed while it flows on the HIGIPS, and the final result is output from stage N.

In preprocessing, data amount to deal with is big, and data transferring time from the global memory to the local memory is longer than other kinds of processing. The bus competition of the global memory accessing will decrease the whole pipeline performance. Hence, data transferring time in the first stage (which usually performs the preprocessing and accesses the global memory most often) is analyzed here. If the bottle-neck does not happen in this stage, it also does not happen in other stages.

6.4.1 Data Transferring Time

To process one frame of image data with M processors, it is necessary to segment the image data into M blocks. For a frame of image data with the size of $K \times L$ bytes, the segmentation method is to divide it into M blocks vertically as shown in Fig.6.13. Each block size is $K \times L/M$ bytes. Each processor transfer not only the corresponding image block but also the δ-line overlapped image data in up- and down-side into its local memory to avoid the image data exchange. Let Π_m represent the image size necessary for processor m to transfer into its local memory, then,

$$\Pi_m = K \times (L/M + \delta \times (\epsilon_1 + \epsilon_2)) \qquad (m=1, 2, \ldots, M) \qquad (6.1)$$

Where $\epsilon_1=\epsilon_2=0$ when $M=1$. And ϵ_1 and ϵ_2 are respectively 0 and 1 when $m=1$, and they are 1 and 0 accordingly when $m=M$. ϵ_1 and ϵ_2 are all 1 in other cases.

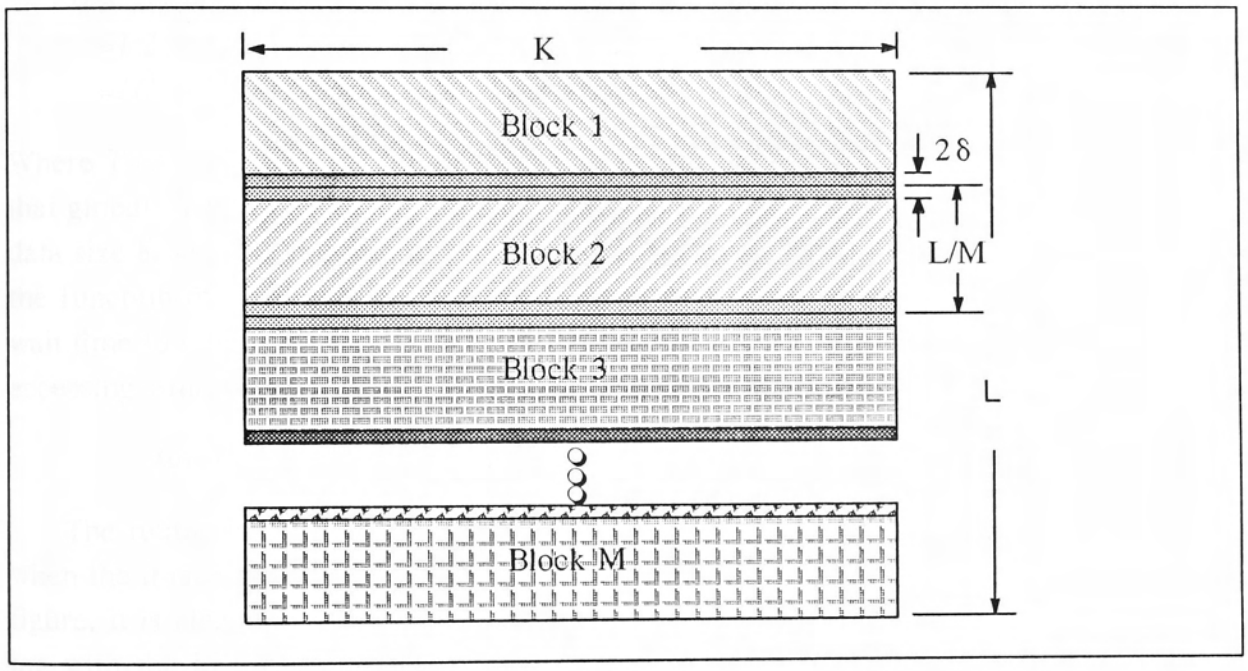

Fig.6.13. Image segmentation into M blocks.

Data transfer from the global memory to the local memory is done by the following instruction.

MOVSW

This is the word block data transfer instruction of the processor (μPD70216). Before running this instruction, it is necessary to put the *source segment* and *source offset* into the segment register **DS** and source index register **SI**, *destination segment* and *destination offset*

into extra segment register **ES** and destination index register **DI**, and the data length (bytes) into the general-purpose register **CX**, respectively. To transfer Π_m bytes image data needs $\Pi_m/2$ repeats of this instruction. Let T_{GAC} represent time for transferring one word from the global memory into the local memory, then,

$$T_{GAC}=11\times T_{CLK} \tag{6.2}$$

In fact, when a processor tries to access the global memory, it must get the priority to use the bus. Let T_{BEN} represent time from the output of bus request signal (of bus controller) to that the bus is enabled, the maximum of T_{BEN} is 747-nS [125].

Let T_G be total time for a processor to transfer one image data block from the global memory into its local memory, and suppose it requests the bus at every time (to transfer one word), then the maximum of T_G can be given as follows.

$$\begin{aligned} T_G &= 1/2\times\Pi_m\times(T_{GAC}+T_{BEN}+T_W) \\ &= 1/2\times K\times(L/M+\delta\times(\epsilon_1+\epsilon_2))\times(11\times 125+747+T_W) \\ &= 1/2\times(2122+T_W)\times K\times(L/M+\delta\times(\epsilon_1+\epsilon_2))\ \text{nS} \end{aligned} \tag{6.3}$$

Where T_W is wait time for the processor to obtain the bus. From this equation, it is clear that global memory accessing time is related to the image size K and L, overlapped image data size δ, and the number of processors (in a stage). When the image size is given, T_G is the function of δ, M and T_W. δ depends upon what kinds of processing to perform. T_W is wait time to obtain the bus. The more the processors, the bigger the T_W is. Global memory accessing time without taking T_W into the consideration is given in the following.

$$T_{GS}=1/2\times 2122\times K\times(L/M+\delta\times(\epsilon_1+\epsilon_2))\ \text{nS} \tag{6.4}$$

The relationship between T_{GS} and the number of the processors is shown in Fig.6.14 when the image size is 320×200 bytes and $\epsilon_1=\epsilon_2=1$ (except when $M=1$, $\epsilon_1=\epsilon_2=0$). From this figure, it is clear that global memory accessing time of each processor decreases according with the increment of the number of the processors because the image size is constant. But if taking T_W also into the consideration, global memory accessing time will be a little bigger.

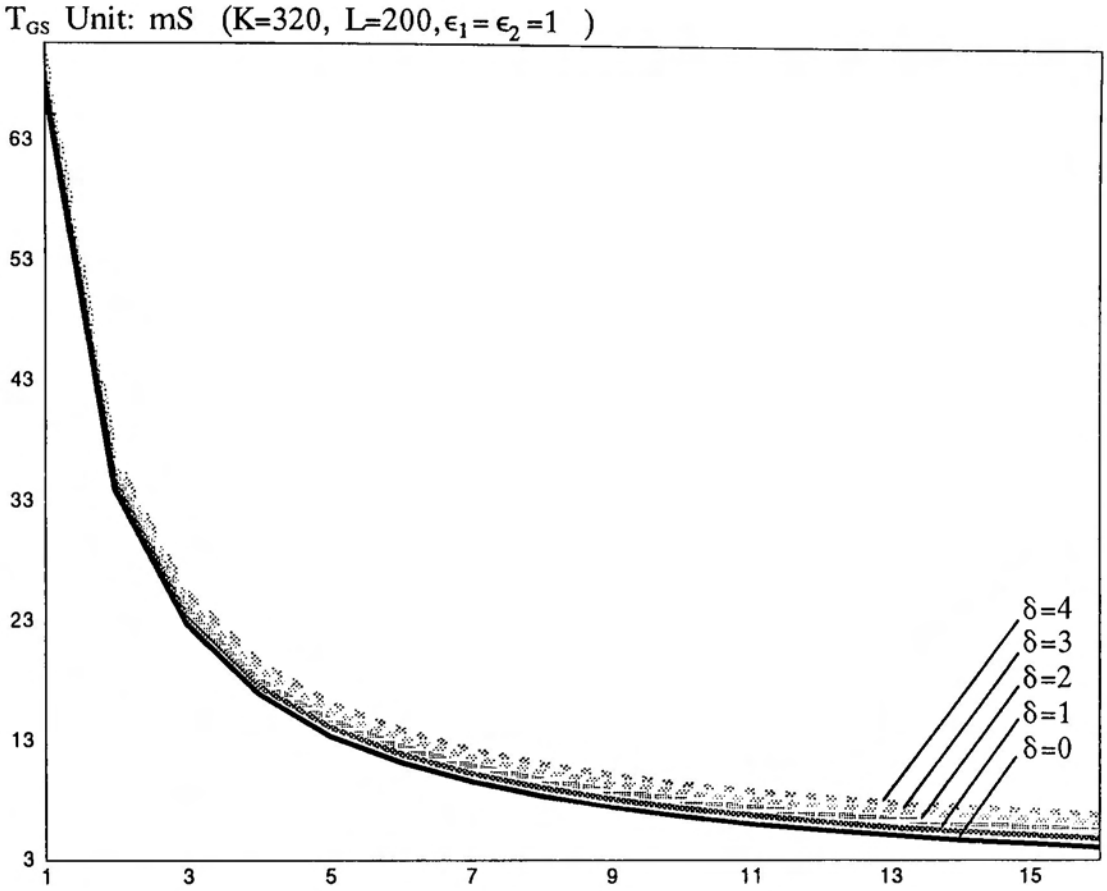

Fig.6.14. Relationship between global memory accessing time and the number of the processors.

6.4.2 Image Processing Time

If an image processing task is given, processing time per pixel is known. Let T_P represent it. To perform this processing by M processors, each processor will deal with the image data block whose size is given by equation (6.1). Since all processors run parallel, necessary time for this task is execution time of one processor. Let T_T represent this time, and suppose this task does not need to process the overlapped image data, then,

$$T_T=(\Pi_m-\delta\times(\epsilon_1+\epsilon_2)\times K)\times T_P$$
$$=K\times L/M\times T_P \quad \text{nS} \tag{6.5}$$

Where if the image size and image processing task are decided, K, L, and T_P are constants, and T_T is the function of M. Fig.6.15 shows the relationship between them in the case that the image size is 320×200 bytes.

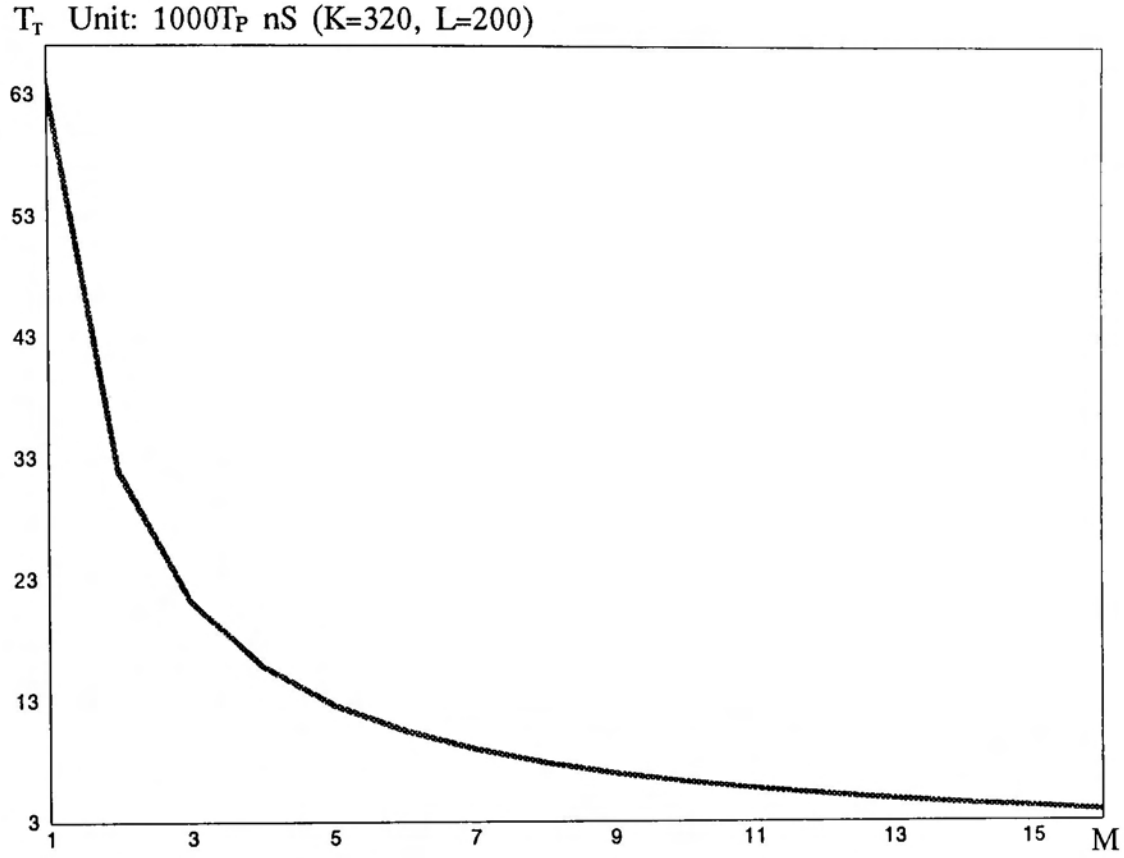

Fig.6.15. Relationship between image processing time and the number of processors.

6.4.3 Global Memory Accessing Rate

The global memory accessing rate is defined as the ratio of global memory accessing time and total processing time. Let R_G represent it, then,

$$\begin{aligned} R_G &= \text{global memory accessing time} \div \text{total processing time} \\ &= T_G/(T_G+T_T) \\ &= 1/(1+(2\times T_P\times L/M)/((2122+T_W)\times(L/M+\delta\times(\epsilon_1+\epsilon_2))) \end{aligned} \tag{6.6}$$

This is used as a criterion to represent the degree of bottle-neck in an image processing task.

6.5 Image Processing Experiments

On the basis of the above analysis, image data transferring and image processing such as (1) thresholding value selection (the calculation of the gray-level histogram and

differential histogram of an assigned image block), (2) sharpening, (3) smoothing, and (4) edge detection (according to Laplacian filter, Sobel operator, Prewitt operator, and Kirsch operator respectively) are done in one stage of HIGIPS with 1-, 2-, ..., 7-processor, respectively. Global memory accessing time and processing time in each case are examined. The programming language is MS-C.

6.5.1 Measurement of Image Data Transferring Time

To examine bus competition time in a stage, experiments for transferring an image block from the global memory to the local memory are done. Each experiment is repeted 10 times, and tthe maximal, minimal and average values are shown in the following.

Table 6.5
Image Data Transferring Speed (Size: 320×20 bytes, Unit: Byte/mS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	543.4	—	—	—	—	—	—
	Max.	543.5	—	—	—	—	—	—
	Min.	543.3	—	—	—	—	—	—
M=2	Ave.	544.7	546.4	—	—	—	—	—
	Max.	545.1	546.4	—	—	—	—	—
	Min.	543.5	546.4	—	—	—	—	—
M=3	Ave.	544.6	414.7	414.8	—	—	—	—
	Max.	544.9	415.6	415.6	—	—	—	—
	Min.	544.2	413.2	413.1	—	—	—	—
M=4	Ave.	543.8	390.5	388.2	379.3	—	—	—
	Max.	544.8	392.6	391.7	385.6	—	—	—
	Min.	542.6	388.4	385.3	376.8	—	—	—
M=5	Ave.	544.3	387.5	386.0	372.7	235.3	—	—
	Max.	544.8	388.9	387.3	377.9	237.1	—	—
	Min.	544.0	386.1	382.1	369.8	233.6	—	—
M=6	Ave.	543.8	386.1	386.1	366.3	210.5	204.4	—
	Max.	543.9	386.5	386.4	367.9	212.1	204.9	—
	Min.	543.7	385.9	385.9	357.4	209.9	203.7	—
M=7	Ave.	540.5	384.0	384.0	367.0	203.8	194.7	189.4
	Max.	541.6	386.0	386.1	382.6	209.1	203.8	202.9
	Min.	539.2	383.1	383.1	364.1	198.9	190.7	186.2

(1) 320×20-byte Image Data Transferring

Transferring an image block with the size of 320×20-byte is performed with 1-, 2-, ..., 7-processor on prototype HIGIPS, respectively. Global memory accessing time is recorded, the average, maximum, and minimum are summarized in table 6.4, the corresponding transferring speeds are calculated and given in table 6.5.

Table 6.4
Image Data Transferring Time (Size: 320×20 bytes, Unit: μS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	11777	—	—	—	—	—	—
	Max.	11780	—	—	—	—	—	—
	Min.	11776	—	—	—	—	—	—
	σ	1	—	—	—	—	—	—
M=2	Ave.	11750	11714	—	—	—	—	—
	Max.	11776	11714	—	—	—	—	—
	Min.	11742	11712	—	—	—	—	—
	σ	9	1	—	—	—	—	—
M=3	Ave.	11753	17434	15429	—	—	—	—
	Max.	11760	15488	15494	—	—	—	—
	Min.	11746	15398	15398	—	—	—	—
	σ	5	32	30	—	—	—	—
M=4	Ave.	11770	16388	16488	16872	—	—	—
	Max.	11794	16478	16612	16984	—	—	—
	Min.	11748	16300	16338	16598	—	—	—
	σ	14	70	101	112	—	—	—
M=5	Ave.	11758	16515	16580	17173	27196	—	—
	Max.	11764	16574	16748	17308	27392	—	—
	Min.	11748	16458	16524	16936	26988	—	—
	σ	6	39	86	132	134	—	—
M=6	Ave.	11768	16574	16575	17473	30397	31310	—
	Max.	11772	16584	16584	17906	30484	31424	—
	Min.	11766	16560	16564	17398	30178	31240	—
	σ	2	8	6	145	115	60	—
M=7	Ave.	11840	16668	16668	17440	31404	32867	33795
	Max.	11870	16704	16704	17580	32182	33564	34380
	Min.	11816	16578	16578	16726	30611	31408	31542
	σ	17	48	48	244	652	546	767

(2) 320×30-byte Image Data Transferring

Transferring an image block with the size of 320×30-byte is performed with 1-, 2-, ..., 7-processor on prototype HIGIPS, respectively. Global memory accessing time is recorded, the average, maximum, and minimum are summarized in table 6.6, the corresponding transferring speeds are calculated and given in table 6.7.

Table 6.7
Image Data Transferring Speed (Size: 320×30 bytes, Unit: Byte/mS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	544.8	—	—	—	—	—	—
	Max.	545.1	—	—	—	—	—	—
	Min.	544.3	—	—	—	—	—	—
M=2	Ave.	545.7	547.3	—	—	—	—	—
	Max.	546.0	547.4	—	—	—	—	—
	Min.	545.3	547.3	—	—	—	—	—
M=3	Ave.	546.0	415.1	414.9	—	—	—	—
	Max.	546.3	415.9	415.8	—	—	—	—
	Min.	545.8	414.0	413.7	—	—	—	—
M=4	Ave.	545.6	389.9	387.1	379.7	—	—	—
	Max.	545.7	393.0	390.3	381.7	—	—	—
	Min.	545.5	388.2	385.3	376.9	—	—	—
M=5	Ave.	544.8	387.0	383.8	372.5	238.3	—	—
	Max.	545.4	387.3	386.8	374.5	240.1	—	—
	Min.	543.4	386.6	381.8	369.8	237.0	—	—
M=6	Ave.	544.5	386.0	385.8	369.3	209.7	204.2	—
	Max.	544.9	386.6	386.5	383.7	213.0	205.0	—
	Min.	544.1	383.7	383.7	367.4	203.9	203.6	—
M=7	Ave.	540.8	382.2	382.2	363.6	201.1	192.3	186.9
	Max.	541.7	383.7	383.7	365.1	205.3	194.2	187.8
	Min.	539.9	378.0	378.0	359.7	197.8	189.8	185.8

Table 6.6

Image Data Transferring Time (Size: 320×30 bytes, Unit: μS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	17622	—	—	—	—	—	—
	Max.	17636	—	—	—	—	—	—
	Min.	17612	—	—	—	—	—	—
	σ	8	—	—	—	—	—	—
M=2	Ave.	17593	17540	—	—	—	—	—
	Max.	17604	17542	—	—	—	—	—
	Min.	17582	17538	—	—	—	—	—
	σ	7	1	—	—	—	—	—
M=3	Ave.	17584	23128	23137	—	—	—	—
	Max.	17588	23190	23208	—	—	—	—
	Min.	17574	23082	23088	—	—	—	—
	σ	4	48	51	—	—	—	—
M=4	Ave.	17596	24624	24798	25284	—	—	—
	Max.	17600	24730	24918	25472	—	—	—
	Min.	17592	24430	24594	25152	—	—	—
	σ	3	109	134	143	—	—	—
M=5	Ave.	17620	24804	25012	25768	40289	—	—
	Max.	17668	24830	25146	25962	40506	—	—
	Min.	17604	24784	24816	25634	39978	—	—
	σ	23	14	156	156	232	—	—
M=6	Ave.	17632	24873	24884	25993	45786	47005	—
	Max.	17644	25022	25022	26132	47086	47146	—
	Min.	17618	24830	24836	25022	45078	46830	—
	σ	8	74	70	324	724	136	—
M=7	Ave.	17751	25120	25120	26400	47737	49915	51356
	Max.	17780	25394	25396	26688	48524	50572	51680
	Min.	17722	25020	25022	26292	46772	49422	51122
	σ	15	127	128	138	697	481	196

(3) 320×40-byte Image Data Transferring

Transferring an image block with the size of 320×40-byte is performed with 1-, 2-, ..., 7-processor on prototype HIGIPS, respectively. Global memory accessing time is recorded, the average, maximum, and minimum are summarized in table 6.8, the corresponding transferring speeds are calculated and given in table 6.9.

Table 6.9
Image Data Transferring Speed (Size: 320×40 bytes, Unit: Byte/mS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	545.4	—	—	—	—	—	—
	Max.	545.6	—	—	—	—	—	—
	Min.	545.3	—	—	—	—	—	—
M=2	Ave.	546.5	547.8	—	—	—	—	—
	Max.	546.6	547.9	—	—	—	—	—
	Min.	546.4	547.8	—	—	—	—	—
M=3	Ave.	546.7	414.2	414.0	—	—	—	—
	Max.	546.7	415.7	415.7	—	—	—	—
	Min.	546.7	413.7	413.4	—	—	—	—
M=4	Ave.	545.2	393.6	389.6	377.0	—	—	—
	Max.	546.1	396.4	392.8	380.1	—	—	—
	Min.	543.7	389.2	383.6	375.5	—	—	—
M=5	Ave.	546.3	384.3	385.1	380.7	234.2	—	—
	Max.	554.6	384.9	387.0	386.8	237.4	—	—
	Min.	544.3	383.9	383.9	369.8	230.6	—	—
M=6	Ave.	544.7	386.7	386.5	369.7	210.1	204.7	—
	Max.	545.1	387.1	387.1	375.7	215.1	205.4	—
	Min.	543.7	386.1	384.0	367.8	207.1	202.9	—
M=7	Ave.	542.7	383.7	383.8	365.3	200.5	193.6	187.5
	Max.	543.0	383.9	383.9	365.5	206.2	194.6	188.1
	Min.	541.4	382.0	383.7	365.0	199.1	190.2	186.0

Table 6.8

Image Data Transferring Time (Size: 320×40 bytes, Unit: μS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	23469	—	—	—	—	—	—
	Max.	23474	—	—	—	—	—	—
	Min.	23462	—	—	—	—	—	—
	σ	4	—	—	—	—	—	—
M=2	Ave.	23422	23365	—	—	—	—	—
	Max.	23428	23368	—	—	—	—	—
	Min.	23416	23362	—	—	—	—	—
	σ	3	2	—	—	—	—	—
M=3	Ave.	23413	30900	30921	—	—	—	—
	Max.	23414	30944	30966	—	—	—	—
	Min.	23412	30794	30794	—	—	—	—
	σ	1	54	64	—	—	—	—
M=4	Ave.	23480	32521	32852	33949	—	—	—
	Max.	23544	32884	33366	34092	—	—	—
	Min.	23440	32292	32588	33674	—	—	—
	σ	41	202	277	165	—	—	—
M=5	Ave.	23429	33309	33234	33618	54646	—	—
	Max.	23518	33344	33342	34614	55500	—	—
	Min.	23080	33252	33072	33090	53916	—	—
	σ	119	33	104	531	710	—	—
M=6	Ave.	23501	33102	33120	34624	60934	62520	—
	Max.	23544	33150	33332	34800	61800	63070	—
	Min.	23482	33070	33070	34074	59508	62332	—
	σ	23	26	74	276	706	283	—
M=7	Ave.	23584	33362	33347	35041	63826	66119	68259
	Max.	23642	33510	33360	35064	64276	67280	68826
	Min.	23572	33340	33342	35016	62086	65788	68056
	σ	21	50	7	13	794	551	268

(4) 320×50-byte Image Data Transferring

Transferring an image block with the size of 320×50-byte is performed with 1-, 2-, ..., 7-processor on prototype HIGIPS, respectively. Global memory accessing time is recorded, the average, maximum, and minimum are summarized in table 6.10, the corresponding transferring rates are calculated and given in table 6.11.

Table 6.11
Image Data Transferring Rate (Size: 320×50 bytes, Unit: Byte/mS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	546.1	—	—	—	—	—	—
	Max.	546.2	—	—	—	—	—	—
	Min.	546.0	—	—	—	—	—	—
M=2	Ave.	531.6	532.7	—	—	—	—	—
	Max.	531.9	532.9	—	—	—	—	—
	Min.	531.3	532.3	—	—	—	—	—
M=3	Ave.	525.4	412.7	421.3	—	—	—	—
	Max.	526.0	414.9	466.3	—	—	—	—
	Min.	524.5	411.9	411.1	—	—	—	—
M=4	Ave.	524.4	390.0	384.8	357.7	—	—	—
	Max.	524.8	392.5	387.2	360.5	—	—	—
	Min.	523.7	386.7	381.7	355.4	—	—	—
M=5	Ave.	523.8	385.4	382.9	352.5	230.5	—	—
	Max.	524.3	387.4	383.7	389.0	234.7	—	—
	Min.	523.6	384.0	382.1	337.6	227.9	—	—
M=6	Ave.	523.8	385.0	383.0	340.2	209.4	201.4	—
	Max.	524.1	385.6	383.6	340.9	211.5	202.8	—
	Min.	523.3	384.7	382.6	339.1	206.9	200.1	—
M=7	Ave.	520.3	384.0	382.1	337.3	200.9	192.1	182.2
	Max.	521.3	384.4	382.3	338.2	205.0	194.0	183.0
	Min.	516.3	383.1	381.8	336.2	198.4	188.8	181.2

Table 6.10

Image Data Transferring Time (Size: 320×50 bytes, Unit: μS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	29299	—	—	—	—	—	—
	Max.	29306	—	—	—	—	—	—
	Min.	29292	—	—	—	—	—	—
	σ	4	—	—	—	—	—	—
M=2	Ave.	30097	30036	—	—	—	—	—
	Max.	30112	30058	—	—	—	—	—
	Min.	30082	30024	—	—	—	—	—
	σ	10	12	—	—	—	—	—
M=3	Ave.	30455	38766	37976	—	—	—	—
	Max.	30508	38846	38924	—	—	—	—
	Min.	30420	38568	34316	—	—	—	—
	σ	35	100	1713	—	—	—	—
M=4	Ave.	30511	41026	44632	44732	—	—	—
	Max.	30552	41374	41916	45020	—	—	—
	Min.	30490	40760	41322	44380	—	—	—
	σ	20	230	243	236	—	—	—
M=5	Ave.	30544	41521	44804	45396	69403	—	—
	Max.	30558	41662	41876	47398	70208	—	—
	Min.	30516	41300	41698	41128	68182	—	—
	σ	16	163	85	1601	981	—	—
M=6	Ave.	30547	41556	41771	47036	76401	79428	—
	Max.	30574	41596	41818	47190	77332	79948	—
	Min.	30530	41498	41712	46934	75654	78890	—
	σ	15	40	45	83	695	429	—
M=7	Ave.	30749	41666	41879	47438	79634	83294	87827
	Max.	30992	41768	41912	47592	80630	84762	88276
	Min.	30694	41620	41848	47316	78054	82492	87454
	σ	84	44	24	111	1105	912	311

(5) 320×100-byte Image Data Transferring

Transferring an image block with the size of 320×100-byte is performed with 1-, 2-, ..., 7-processor on prototype HIGIPS, respectively. Global memory accessing time is recorded, the average, maximum, and minimum are summarized in table 6.12, the corresponding transferring speeds are calculated and given in table 6.13.

Table 6.13
Image Data Transferring Rate (Size: 320×100 bytes, Unit: Byte/mS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	546.7	—	—	—	—	—	—
	Max.	546.8	—	—	—	—	—	—
	Min.	546.6	—	—	—	—	—	—
M=2	Ave.	472.9	474.0	—	—	—	—	—
	Max.	473.5	474.9	—	—	—	—	—
	Min.	470.8	471.3	—	—	—	—	—
M=3	Ave.	447.7	399.1	394.0	—	—	—	—
	Max.	448.8	401.1	398.1	—	—	—	—
	Min.	444.7	398.2	393.2	—	—	—	—
M=4	Ave.	446.5	388.4	377.1	286.2	—	—	—
	Max.	448.4	390.1	378.9	287.8	—	—	—
	Min.	445.8	386.6	376.1	285.1	—	—	—
M=5	Ave.	445.6	387.5	376.8	264.6	218.3	—	—
	Max.	445.9	387.8	377.0	265.9	219.4	—	—
	Min.	445.0	386.9	376.2	262.7	217.6	—	—
M=6	Ave.	445.7	386.8	378.5	258.1	202.1	196.0	—
	Max.	446.1	387.7	385.9	260.5	203.9	197.1	—
	Min.	445.2	385.9	376.0	254.3	201.5	195.6	—
M=7	Ave.	444.8	384.9	375.0	256.3	151.4	109.8	108.6
	Max.	446.0	385.7	375.7	257.4	151.4	110.3	108.7
	Min.	442.9	384.1	373.3	252.0	151.4	109.6	108.5

Table 6.12

Image Data Transferring Time (Size: 320×100 bytes, Unit: μS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	58533	—	—	—	—	—	—
	Max.	58542	—	—	—	—	—	—
	Min.	58524	—	—	—	—	—	—
	σ	5	—	—	—	—	—	—
M=2	Ave.	67666	67512	—	—	—	—	—
	Max.	67964	67894	—	—	—	—	—
	Min.	67586	67388	—	—	—	—	—
	σ	144	131	—	—	—	—	—
M=3	Ave.	71480	80182	81228	—	—	—	—
	Max.	71962	80354	81392	—	—	—	—
	Min.	71296	79788	80382	—	—	—	—
	σ	250	284	284	—	—	—	—
M=4	Ave.	71675	82381	84851	111823	—	—	—
	Max.	71780	82770	85084	112230	—	—	—
	Min.	71362	82022	84458	111200	—	—	—
	σ	118	260	223	342	—	—	—
M=5	Ave.	71815	82585	84936	120936	146600	—	—
	Max.	71912	82716	85060	121800	147088	—	—
	Min.	71762	82520	84872	120348	145882	—	—
	σ	60	75	65	506	443	—	—
M=6	Ave.	71790	82730	84541	124004	158311	163237	—
	Max.	71884	82920	85096	125826	158820	163640	—
	Min.	71728	82530	82918	122828	156944	162392	—
	σ	61	157	821	1027	691	434	—
M=7	Ave.	71942	83129	85340	124877	211394	291356	294573
	Max.	72246	83318	85716	126970	211400	291938	295038
	Min.	71756	82972	85180	124336	211392	290206	294386
	σ	137	103	160	989	3	697	182

(6) 320×200-byte Image Data Transferring

Transferring an image block with the size of 320×200-byte is performed with 1-, 2-, ..., 7-processor on prototype HIGIPS, respectively. Global memory accessing time is recorded, the average, maximum, and minimum are summarized in table 6.14, the corresponding transferring speeds are calculated and given in table 6.15.

Table 6.15
Image Data Transferring Rate (Size: 320×200 bytes, Unit: Byte/mS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	546.9	—	—	—	—	—	—
	Max.	547.1	—	—	—	—	—	—
	Min.	545.8	—	—	—	—	—	—
M=2	Ave.	448.0	448.4	—	—	—	—	—
	Max.	448.8	449.4	—	—	—	—	—
	Min.	445.6	445.7	—	—	—	—	—
M=3	Ave.	418.0	395.4	392.0	—	—	—	—
	Max.	448.8	399.5	393.4	—	—	—	—
	Min.	445.6	394.6	388.8	—	—	—	—
M=4	Ave.	414.0	386.1	375.9	259.4	—	—	—
	Max.	415.4	387.6	384.9	261.1	—	—	—
	Min.	412.1	382.5	372.6	257.1	—	—	—
M=5	Ave.	411.9	384.6	370.2	234.2	209.8	—	—
	Max.	412.0	385.3	371.1	237.6	210.7	—	—
	Min.	411.8	384.5	369.9	233.2	205.9	—	—
M=6	Ave.	412.7	385.6	372.4	227.0	195.9	193.1	—
	Max.	413.2	386.8	384.1	229.8	198.5	194.8	—
	Min.	412.3	385.1	370.7	220.7	186.0	192.2	—
M=7	Ave.	410.4	384.9	378.3	220.1	147.4	109.4	95.3
	Max.	419.5	385.1	386.3	235.8	159.3	113.1	96.0
	Min.	385.0	384.7	370.5	210.5	138.8	105.9	94.8

Table 6.14

Image Data Transferring Time (Size: 320×200 bytes, Unit: μS)

		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	Ave.	117024	—	—	—	—	—	—
	Max.	117268	—	—	—	—	—	—
	Min.	116974	—	—	—	—	—	—
	σ	82	—	—	—	—	—	—
M=2	Ave.	142864	142733	—	—	—	—	—
	Max.	143642	143610	—	—	—	—	—
	Min.	142594	142410	—	—	—	—	—
	σ	381	434	—	—	—	—	—
M=3	Ave.	153104	161845	163270	—	—	—	—
	Max.	154944	162182	164630	—	—	—	—
	Min.	152338	160190	162676	—	—	—	—
	σ	987	592	592	—	—	—	—
M=4	Ave.	154572	165741	170268	246752	—	—	—
	Max.	155300	167314	171766	248920	—	—	—
	Min.	154066	165108	166264	245080	—	—	—
	σ	378	794	1685	1453	—	—	—
M=5	Ave.	155393	166409	172867	273227	304966	—	—
	Max.	155412	166452	173000	274434	310762	—	—
	Min.	155354	166090	172454	269376	303740	—	—
	σ	18	107	141	15533	2011	—	—
M=6	Ave.	155065	165968	171853	281881	326639	331476	—
	Max.	155228	166208	172654	289982	344158	332944	—
	Min.	154870	165470	166638	278520	322482	328536	—
	σ	130	218	1744	3545	6044	1321	—
M=7	Ave.	155954	166292	169196	290834	303207	322934	409471
	Max.	166232	166366	172732	303974	330054	342348	413114
	Min.	152558	166196	165672	271438	270802	303686	404778
	σ	3521	76	3490	13393	26288	18752	3730

According to table 6.7, 6.9, 6,11, 6.13, 6.15, the data transferring speed of every processor in each case is compared and shown in Fig.6.16 to Fig.6.22. For SUBC, the data transferring speed does not change when the image data size is less than 320×40 bytes (see Fig.6.16). For SMPU1 and SMPU2, the data transferring speed is around 400 byte/mS and 385 byte/mS respectively, and does not change dramatically when the number of processors is increased or a larger size image block is transferred (see Fig.6.17 and Fig.6.18). The transferring speed of SMPU3 is around 370 byte/mS when the image size is less than 320×50 bytes (see Fig.6.19). For SMPU4, SMPU5, and SMPU6, the data transferring speed will decrease greatly when the number of processor is increased (see Fig.6.20, Fig.6.21, and Fig.6.22).

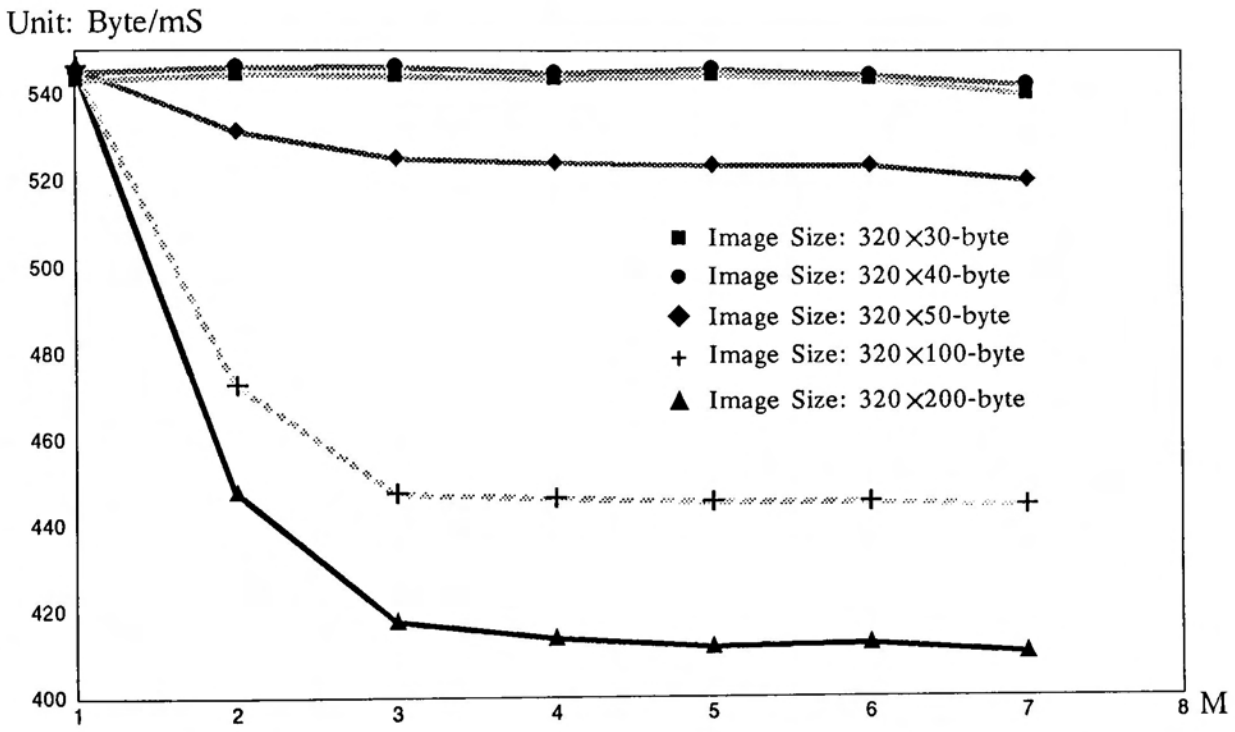

Fig.6.16. Changes of average image data transferring speed of SUBC.

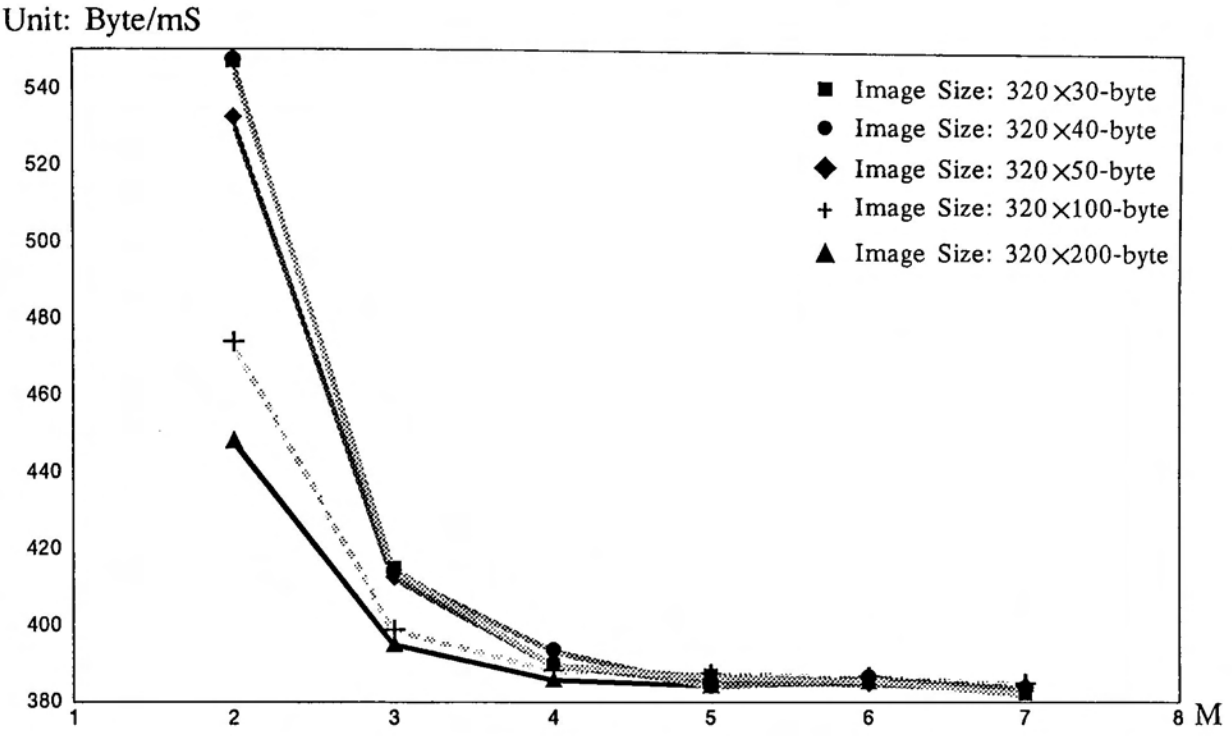

Fig.6.17. Changes of average image data transferring speed of SMPU1.

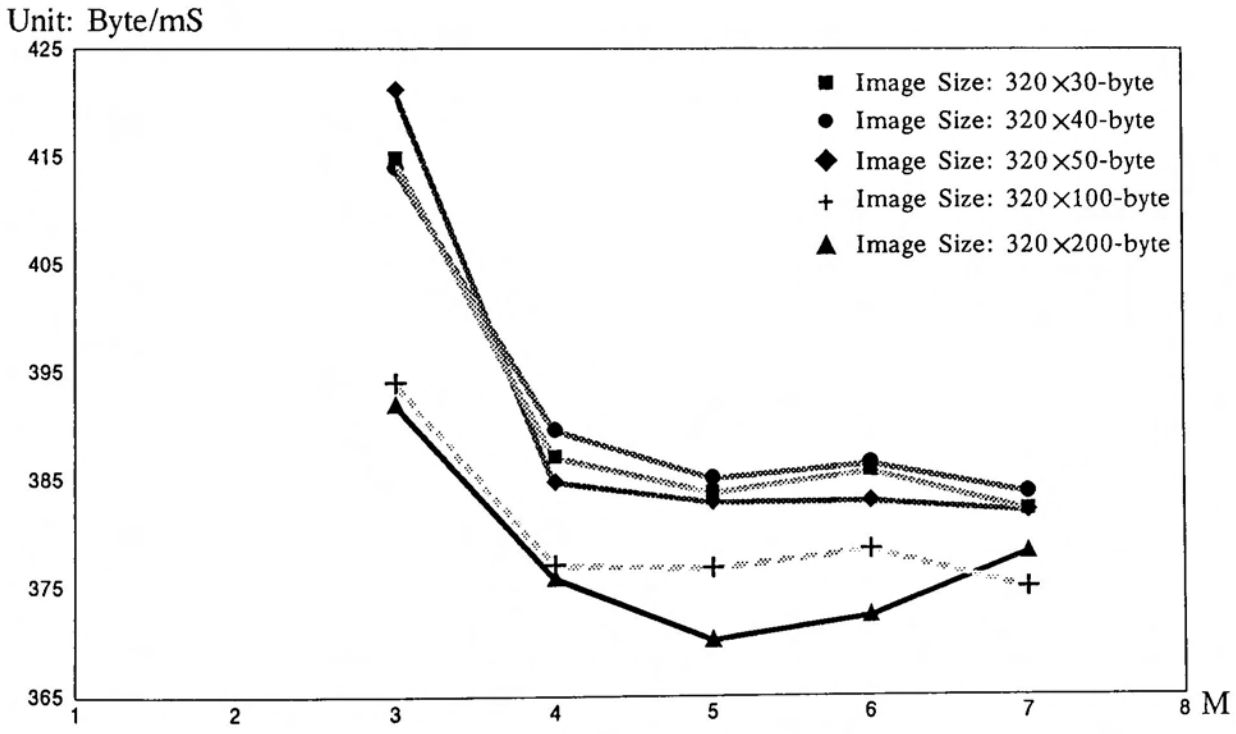

Fig.6.18. Changes of image data transferring rate of SMPU2.

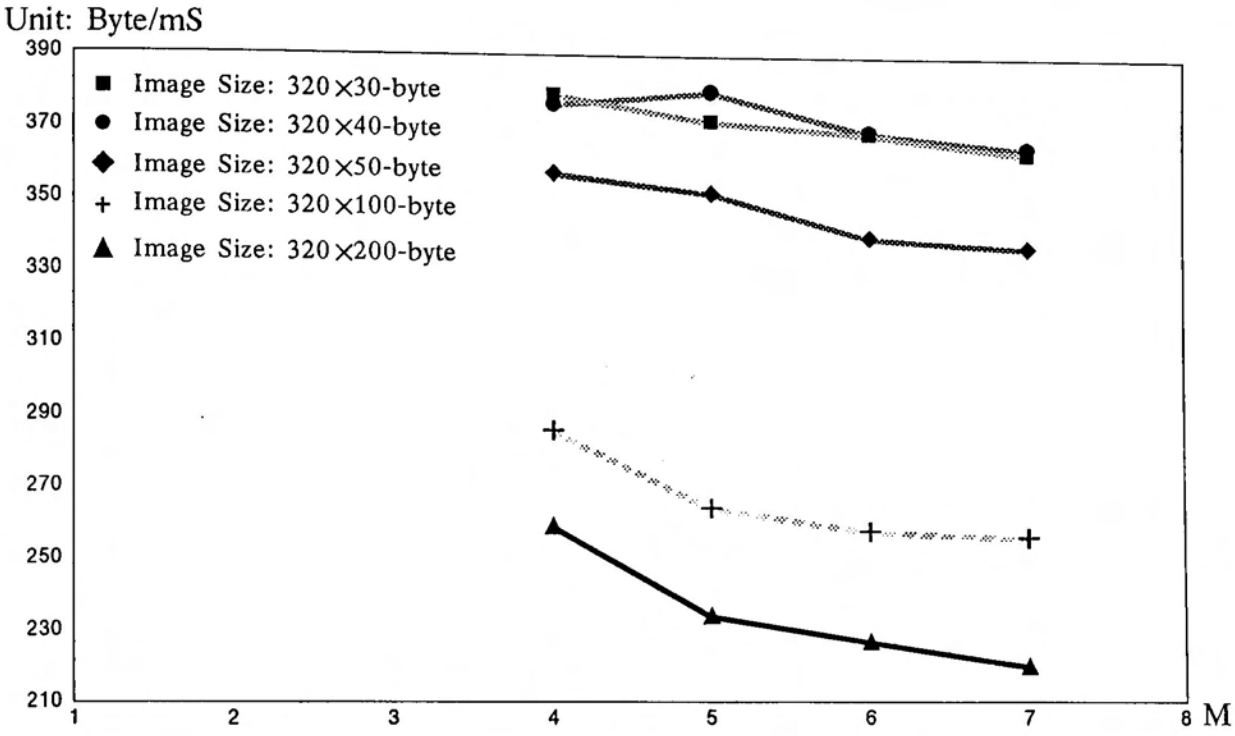

Fig.6.19. Changes of average image data transferring speed of SMPU3.

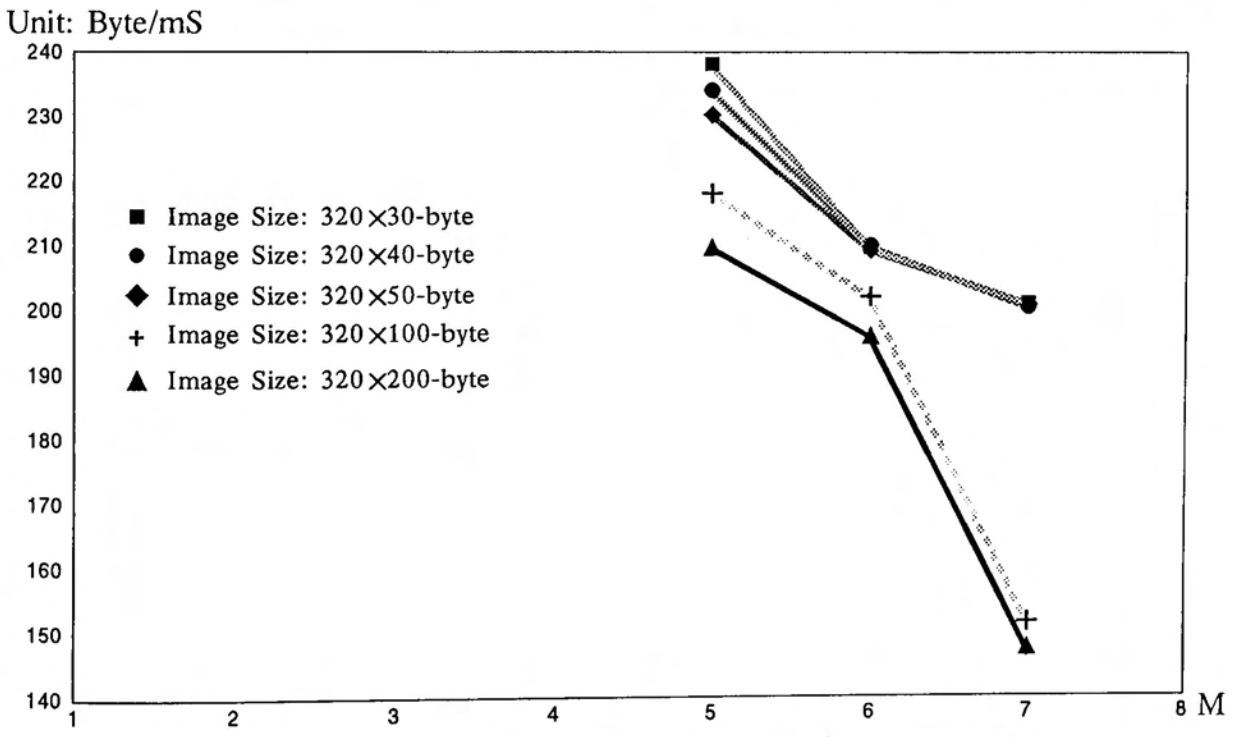

Fig.6.20. Changes of average image data transferring speed of SMPU4.

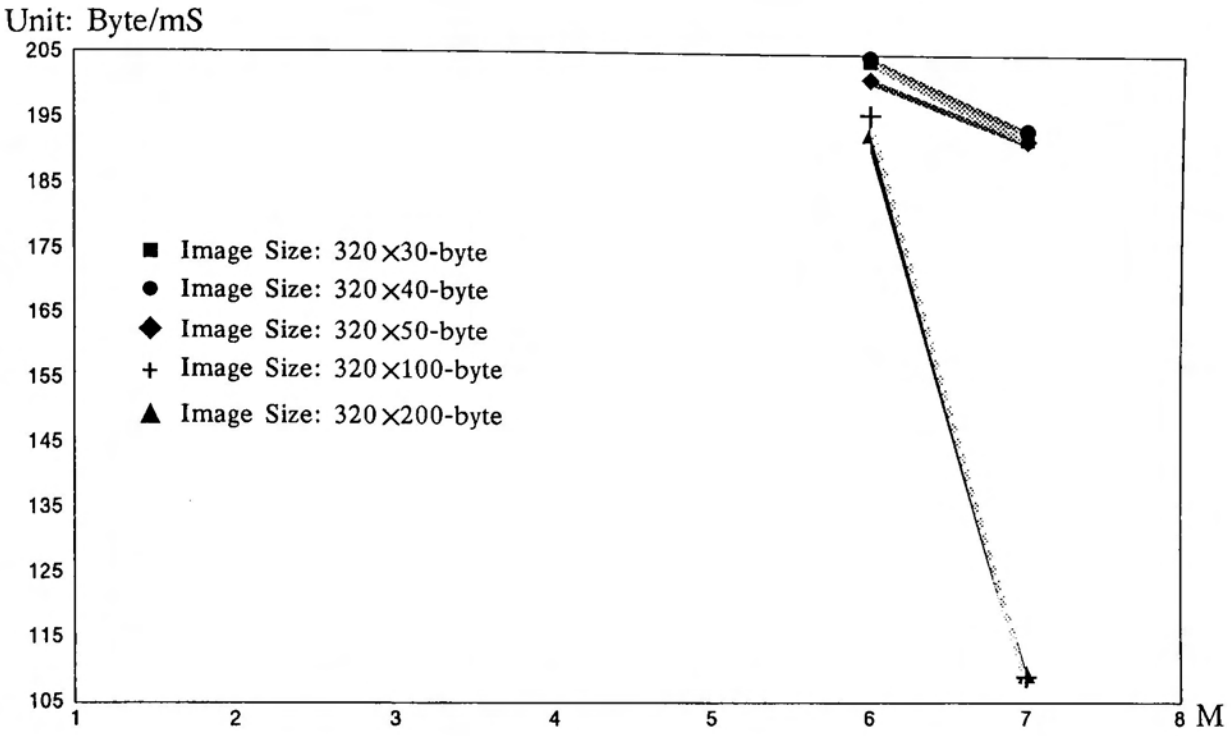

Fig.6.21. Changes of average image data transferring speed of SMPU5.

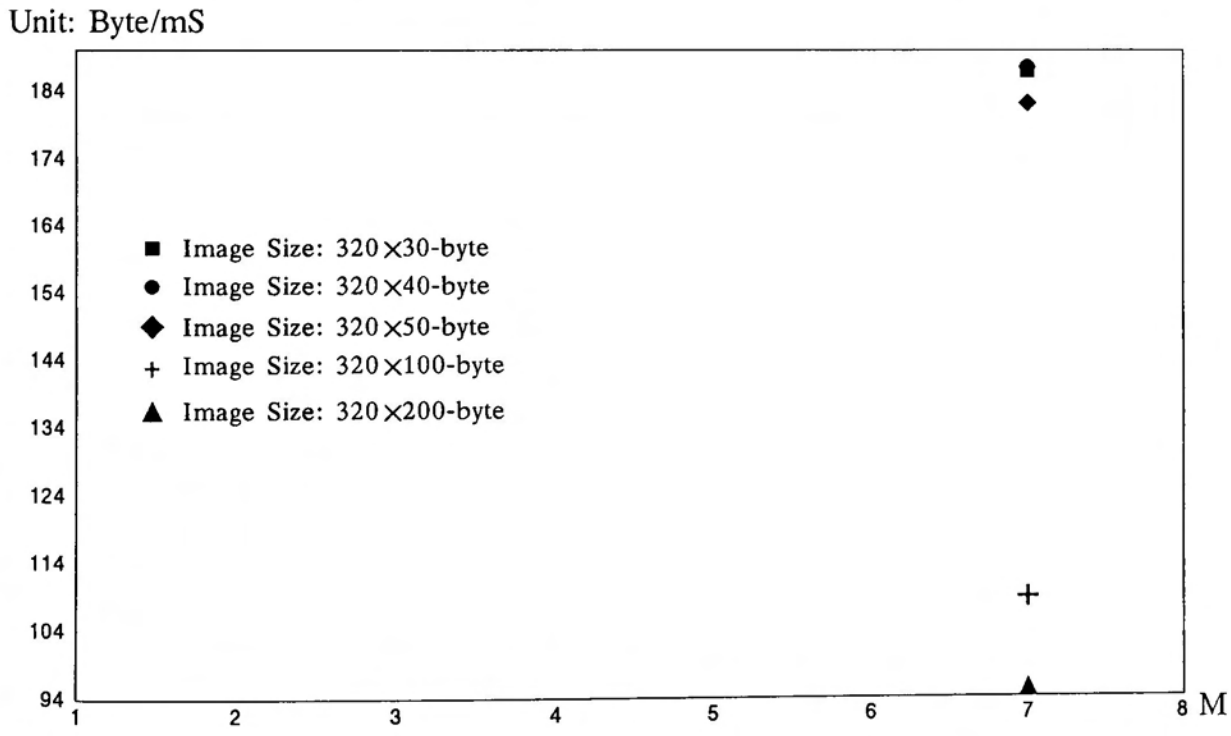

Fig.6.22. Changes of average image data transferring speed of SMPU6.

6.5.2 Measurement of Image Processing Time

Four kinds of image processing experiments are done on the prototype HIGIPS, and their processing time is recorded. From these recorded data, global memory accessing rate and the speed up in each case is calculated. The flowing shows the details.

These processing operations are performed on the first stage of the prototype HIGIPS. Each is performed in seven cases. In case *1*, *M* equals to *1*, and the processing stage includes only one CPU. In case *2*, *M* equals to *2*, and the processing stage includes two CPU's. Likewise, In case *7*, *M* equals to *7*, and the processing stage include seven CPU's. In case one, 320×200-byte input image is stored into M11 of this processing stage. In case *2* to *7*, the input image is divided according to equation (6.1). All processors access their corresponding region, perform the assigned processing, and output the results to M22 (refer to Fig.3.15). ALL processors run at 8-MHz and the common bus clock is 10-MHz.

Table 6.16

T_G, T_T, and R_G of Calculation of Gray-level Histogram (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	2048	—	—	—	—	—	—
bytes	R_G	5.40%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	1024	1024	—	—	—	—	—
bytes	R_G	6.23%	6.23%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	697	697	697	—	—	—	—
bytes	R_G	6.06%	7.19%	7.19%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	528	528	528	528	—	—	—
bytes	R_G	5.71%	7.37%	7.53%	8.17%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	422	422	422	422	422	—	—
bytes	R_G	5.59%	7.66%	7.66%	7.66%	12.1%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	348	348	348	348	348	348	—
bytes	R_G	5.69%	7.69%	7.69%	7.94%	12.8%	13.4%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	296	296	296	296	296	296	296
bytes	R_G	5.73%	7.79%	7.79%	8.07%	13.7%	14.5%	14.7%

(1) Threshold Selection

Thresholding value can be obtained according to gray-level histogram or differential histogram. They are described as follows.

A. Calculation of Gray-level Histogram

If there exist two peaks (corresponding to object and background) in gray-level histogram, the gray-level value corresponding to the ravine between peaks can be selected as thresholding value. This calculation is performed by 1-, 2-, ..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.16.

Table 6.17

T_G, T_T, and R_G of Calculation of Differential Histogram (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	19904	—	—	—	—	—	—
bytes	R_G	0.58%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	9952	9952	—	—	—	—	—
bytes	R_G	0.68%	0.68%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	6991	6991	6991	—	—	—	—
bytes	R_G	0.64%	0.77%	0.77%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	4976	4976	4976	4976	—	—	—
bytes	R_G	0.64%	0.84%	0.86%	0.94%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	3981	3981	3981	3981	3981	—	—
bytes	R_G	0.63%	0.87%	0.87%	0.87%	1.44%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	3495	3495	3495	3495	3495	3495	—
bytes	R_G	0.60%	0.82%	0.82%	0.85%	1.44%	1.52%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	2966	2966	2966	2966	2966	2966	2966
bytes	R_G	0.60%	0.84%	0.84%	0.87%	1.56%	1.66%	1.69%

B. Calculation of Differential Histogram

If the image is too complicated or includes too much noise, there does not exist a ravine clearly in gray-level histogram. In this case, differential histogram is applied to select thresholding value. The gray-level value in which the differential histogram has the maximum can be selected as thresholding value. This calculation is performed by 1-, 2-, ..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.17.

Table 6.18

T_G, T_T, and R_G of Sharpening Operation (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	4288	—	—	—	—	—	—
bytes	R_G	2.66%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	2144	2144	—	—	—	—	—
bytes	R_G	3.07%	3.07%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	1415	1415	1415	—	—	—	—
bytes	R_G	3.08%	3.68%	3.68%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	1072	1072	1072	1072	—	—	—
bytes	R_G	2.90%	3.77%	3.86%	4.20%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	858	858	858	858	858	—	—
bytes	R_G	2.83%	3.92%	3.92%	3.92%	6.33%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	708	708	708	708	708	708	—
bytes	R_G	2.88%	3.93%	3.93%	4.07%	6.72%	7.09%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	600	600	600	600	600	600	600
bytes	R_G	2.91%	4.00%	4.00%	4.00%	7.26%	7.69%	7.83%

(2) Sharpening

Image sharpening is done by the following equation.

$$f_{SP}(i,j)=5\times f(i,j)-[f(i+1,j)+f(i-1,j)+f(i,j+1)+f(i,j-1)]$$

where $f_{SP}(i,j)$ is the output image after sharpening. This operation is performed by 1-, 2-,

..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.18.

Table 6.19
T_G, T_T, and R_G of Smoothing Operation (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	23424	—	—	—	—	—	—
bytes	R_G	0.50%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	11712	11712	—	—	—	—	—
bytes	R_G	0.58%	0.58%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	7712	7712	7712	—	—	—	—
bytes	R_G	0.58%	0.69%	0.69%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	5856	5856	5856	5856	—	—	—
bytes	R_G	0.54%	0.71%	0.73%	0.80%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	4685	4685	4685	4685	4685	—	—
bytes	R_G	0.53%	0.74%	0.74%	0.74%	1.22%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	3865	3865	3865	3865	3865	3865	—
bytes	R_G	0.54%	0.74%	0.74%	0.77%	1.30%	1.38%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	3279	3279	3279	3279	3279	3279	3279
bytes	R_G	0.55%	0.76%	0.76%	0.79%	1.41%	1.50%	1.53%

(3) Smoothing

Image smoothing is performed by median filtering. That is, if using a 3×3 mask, the output value is the middle one when arranging these nine data from to small to big. It is performed by 1-, 2-, ..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.19.

(4) Edge Detection

Edge detection can be performed by Laplacian filter, Sobel operator, Prewitt operator, or Kirsch operator.

A. Laplacian Filtering

Laplacian filtering is shown by the following equation.

$$f_L(i,j)=f(i+1,j)+f(i-1,j)+f(i,j+1)+f(i,j-1)-4\times f(i,j)$$

where $f_L(i,j)$ is the output image after Laplacian filtering. Laplacian filtering is performed by 1-, 2-, ..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.20.

Table 6.20

T_G, T_T, and R_G of Laplacian Filtering (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	4557	—	—	—	—	—	—
bytes	R_G	2.50%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	2278	2278	—	—	—	—	—
bytes	R_G	2.90%	2.90%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	1504	1504	1504	—	—	—	—
bytes	R_G	2.91%	3.47%	3.48%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	1139	1139	1139	1139	—	—	—
bytes	R_G	2.73%	3.56%	3.64%	3.96%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	911	911	911	911	911	—	—
bytes	R_G	2.67%	3.70%	3.70%	3.70%	5.99%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	752	752	752	752	752	752	—
bytes	R_G	2.72%	3.71%	3.71%	3.84%	6.35%	6.70%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	638	638	638	638	638	638	638
bytes	R_G	2.74%	3.77%	3.77%	3.92%	6.86%	7.27%	7.40%

B. Sobel Operator

Sobel operator is shown by the following equations.

$$f_S(i,j)=|\Delta_x f(i,j)|+|\Delta_y f(i,j)|$$

where $\Delta_x f(i,j)\equiv f(i-1,j-1)+2\times f(i-1,j)+f(i-1,j+1)-f(i+1,j-1)-2\times f(i+1,j)-f(i+1,j+1);$

$$\Delta_y f(i,j)\equiv f(i-1,j-1)+2\times f(i,j-1)+f(i+1,j-1)-f(i-1,j+1)-2\times f(i,j+1)-f(i+1,j+1)$$

and $f_S(i,j)$ is the output image after Sobel operator processing. Sobel operator processing is performed by 1-, 2-, ..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.21.

Table 6.21

T_G, T_T, and R_G of Sobel Operator (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	9958	—	—	—	—	—	—
bytes	R_G	1.16%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	4979	4979	—	—	—	—	—
bytes	R_G	1.35%	1.35%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	3286	3286	3286	—	—	—	—
bytes	R_G	1.35%	1.62%	1.62%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	2490	2490	2490	2490	—	—	—
bytes	R_G	1.23%	1.66%	1.70%	1.85%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	1992	1992	1992	1992	1992	—	—
bytes	R_G	1.24%	1.23%	1.23%	1.23%	2.83%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	1643	1643	1643	1643	1643	1643	—
bytes	R_G	1.26%	1.73%	1.73%	1.79%	3.01%	3.18%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	1394	1394	1394	1394	1394	1394	1394
bytes	R_G	1.27%	1.76%	1.76%	1.83%	3.26%	3.46%	3.53%

C. Prewitt Operator

Prewitt operator is shown by the following equations.

$$f_P(i,j)=|\Delta_x f(i,j)|+|\Delta_y f(i,j)|$$

where $\Delta_x f(i,j)\equiv f(i-1,j-1)+f(i-1,j)+f(i-1,j+1)-f(i+1,j-1)-f(i+1,j)-f(i+1,j+1)$;

$$\Delta_y f(i,j)\equiv f(i-1,j-1)+f(i,j-1)+f(i+1,j-1)-f(i-1,j+1)-f(i,j+1)-f(i+1,j+1)$$

and $f_P(i,j)$ is the output image after Prewitt operator processing. Prewitt operator processing is performed by 1-, 2-, ..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.22.

Table 6.22

T_G, T_T, and R_G of Prewitt Operator (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	9933	—	—	—	—	—	—
bytes	R_G	1.16%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	4966	4966	—	—	—	—	—
bytes	R_G	1.35%	1.35%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	3279	3279	3279	—	—	—	—
bytes	R_G	1.35%	1.62%	1.62%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	2483	2483	2483	2483	—	—	—
bytes	R_G	1.27%	1.66%	1.70%	1.86%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	1987	1987	1987	1987	1987	—	—
bytes	R_G	1.19%	1.73%	1.73%	1.73%	2.84%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	1639	1639	1639	1639	1639	1639	—
bytes	R_G	1.27%	1.74%	1.74%	1.80%	3.02%	3.19%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	1391	1391	1391	1391	1391	1391	1391
bytes	R_G	1.28%	1.77%	1.77%	1.83%	3.27%	3.47%	3.54%

D. Kirsch Operator

Kirsch operator is shown by the following equations.

$$f_K(i,j)=max\{M0,M1,M2,M3,M4,M5,M6,M7\}$$

where *M0, ..., M7* are weighting matrices shown in Fig.6.22, and $f_K(i,j)$ is the output image after Kirsch operator processing. Kirsch operator processing is performed by 1-, 2-, ..., 7-processor respectively. Processing time, data transferring time for the corresponding block, and the global memory accessing rate in each case are summarized in table 6.23.

Table 6.23

T_G, T_T, and R_G of Kirsch Operator (Unit: mS)

Image Size		SUBC	SMPU1	SMPU2	SMPU3	SMPU4	SMPU5	SMPU6
M=1	T_G	117	—	—	—	—	—	—
320×200	T_T	20928	—	—	—	—	—	—
bytes	R_G	0.56%	—	—	—	—	—	—
M=2	T_G	68	68	—	—	—	—	—
320×101	T_T	10464	10464	—	—	—	—	—
bytes	R_G	0.65%	0.65%	—	—	—	—	—
M=3	T_G	45	54	54	—	—	—	—
320×68	T_T	6906	6906	6906	—	—	—	—
bytes	R_G	0.65%	0.78%	0.78%	—	—	—	—
M=4	T_G	32	42	43	47	—	—	—
320×52	T_T	5232	5232	5232	5232	—	—	—
bytes	R_G	0.61%	0.80%	0.82%	0.89%	—	—	—
M=5	T_G	25	35	35	35	58	—	—
320×42	T_T	4186	4186	4186	4186	4186	—	—
bytes	R_G	0.59%	0.83%	0.83%	0.83%	1.37%	—	—
M=6	T_G	21	29	29	30	51	54	—
320×35	T_T	3453	3453	3453	3453	3453	3453	—
bytes	R_G	0.60%	0.83%	0.83%	0.86%	1.46%	1.54%	—
M=7	T_G	18	25	25	26	47	50	51
320×30	T_T	2930	2930	2930	2930	2930	2930	2930
bytes	R_G	0.61%	0.85%	0.85%	0.88%	1.58%	1.68%	1.71%

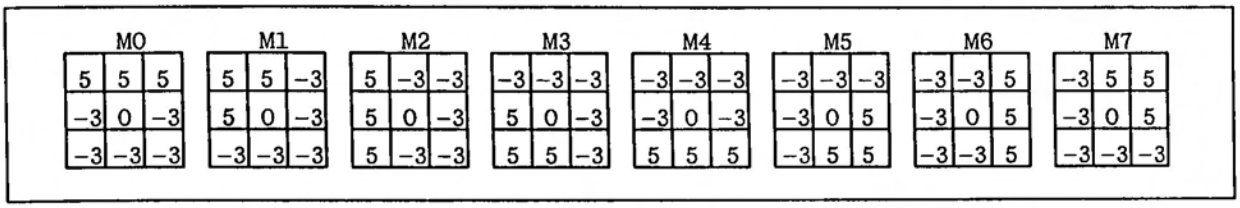

Fig.6.23. Kirsch operator.

From table 6.16 to table 6.23, T_G, T_T, and R_G of the slowest processor (which decides processing time in that stage) for every processing in each case are summarized in table 6.24. From this it is clear that the global memory accessing does not exceed 10% (except the very simple processing such as the calculation of the gray-level histogram). Therefore, there does not exist bottle-neck in the most often global memory accessing stage. Fig.6.24 shows the decrement of processing time of 4 kinds of processing operations with the increment of the processors, and Fig.6.25 shows the changes of the global memory accessing rate of them.

The degree of speed up in each case can be defined as follows:

total time in case of *1*-processor ÷ total time in case of *M*-processor

$=(T_G+T_T)$ in case of *1*-processor ÷ (T_G+T_T) in case of *M*-processor (*M=2, 3, ..., 7*)

The degree of speed up for every processing in each case is summarized in table 6.25. For any processing, around *M*-time speed up is obtained. With these data, a graph is drawn in Fig.6.26 to show the degree of speed up when the number of processor is increased.

In above, we discussed the bus competition in the worst case, i.e., at the beginning, each processor transfers the image block assigned to it from the global memory into its local memory. During processing, it will not access the global memory any more until it places the intermediate result (or final result) back to the global memory. But it is possible for each processor to transfer a small size of image block (e.g., several lines) to its local memory at very beginning, and then start the processing. The left data will be transferred at any time when necessary. This approach will decrease the global memory accessing rate.

And in fact, the processing task is usually complicated than the ones related here, in that case, the global memory accessing rate will be much more smaller. Therefore, there does not exist bottle-neck phenomenon when applying HIGIPS to the field of image processing and analysis.

Table 6.24

Maximal T_G, T_T, and R_G of Every Processing (Unit: mS)

Processing Operations		Max.	M=1	M=2	M=3	M=4	M=5	M=6	M=7
Selection of Thres. Value	Gray-Level Histogram $T_P=256T_{CLK}$, $\delta=0$	T_G	117	68	54	47	58	54	51
		T_T	2048	1024	697	528	422	348	296
		R_G	5.40%	6.23%	7.19%	8.17%	12.1%	13.4%	14.6%
	Differential Histogram $T_P=2488T_{CLK}$, $\delta=0$	T_G	117	68	54	47	58	54	51
		T_T 19904	9952	6991	4976	3981	3495	2966	
		R_G	0.58%	0.68%	0.77%	0.94%	1.44%	1.52%	1.69%
	Sharpening $T_P=536T_{CLK}$ $\delta=1$	T_G	117	68	54	47	58	54	51
		T_T	4288	2144	1415	1072	858	708	600
		R_G	2.66%	3.07%	3.68%	4.20%	6.33%	7.09	7.83%
	Smoothing $T_P=2928T_{CLK}$ $\delta=1$	T_G	117	68	54	47	58	54	51
		T_T	23424	11712	7729	5856	4685	3865	3279
		R_G	0.50%	0.58%	0.69%	0.80%	1.22%	1.38%	1.53%
	Laplacian Filter $T_P=570T_{CLK}$, $\delta=1$	T_G	117	68	54	47	58	54	51
		T_T	4557	2278	1504	1139	911	752	638
		R_G	2.50%	2.90%	3.48%	3.96%	5.99%	6.70%	7.40%
Edge Extraction	Sobel Operator $T_P=1195T_{CLK}$, $\delta=1$	T_G	117	68	54	47	58	54	51
		T_T	9558	4979	3286	2490	1992	1643	1394
		R_G	1.16%	1.35%	1.62%	1.85%	2.83%	3.18%	3.53%
	Prewitt Operator $T_P=1242T_{CLK}$, $\delta=1$	T_G	117	68	54	47	58	54	51
		T_T	9933	4966	3279	2483	1987	1639	1391
		R_G	1.16%	1.35%	1.62%	1.85%	2.84%	3.19%	3.54%
	Kirsch Operator $T_P=2616T_{CLK}$, $\delta=1$	T_G	117	68	54	47	58	54	51
		T_T	20928	10464	6906	5232	4186	3453	2930
		R_G	0.56%	0.65%	0.78%	0.89%	1.37%	1.54%	1.71%

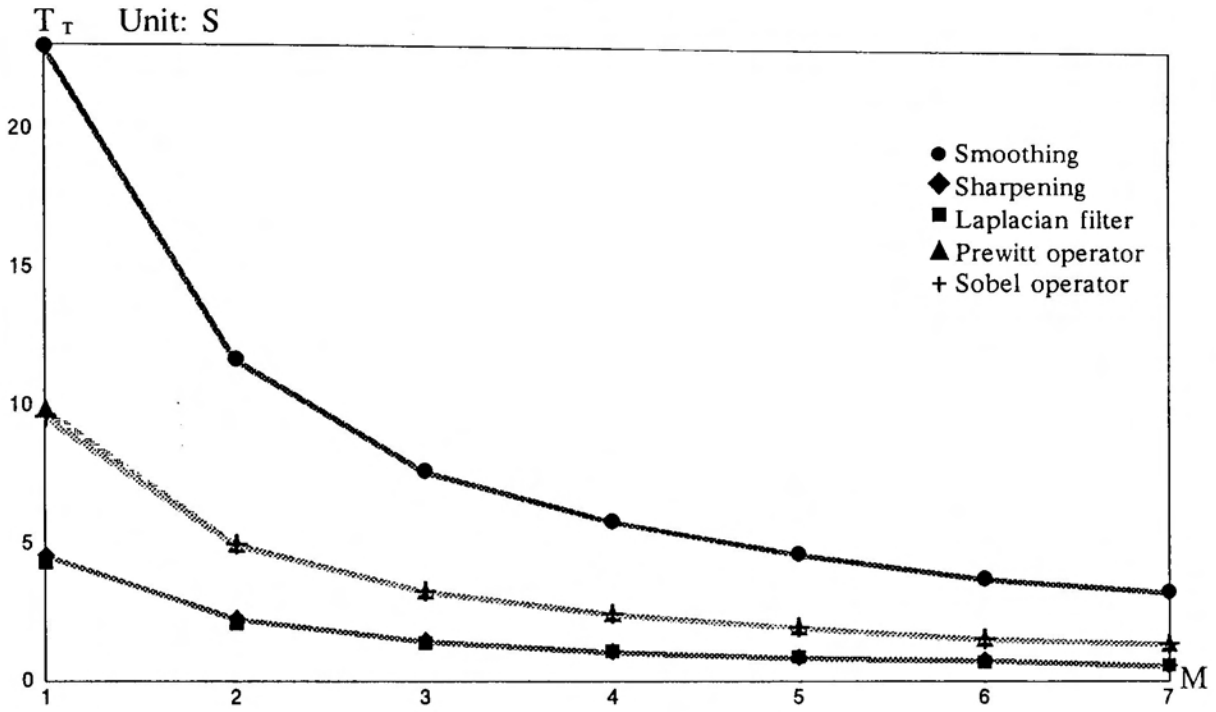

Fig.6.24. Relationship between experimental image processing time and the number of processors.

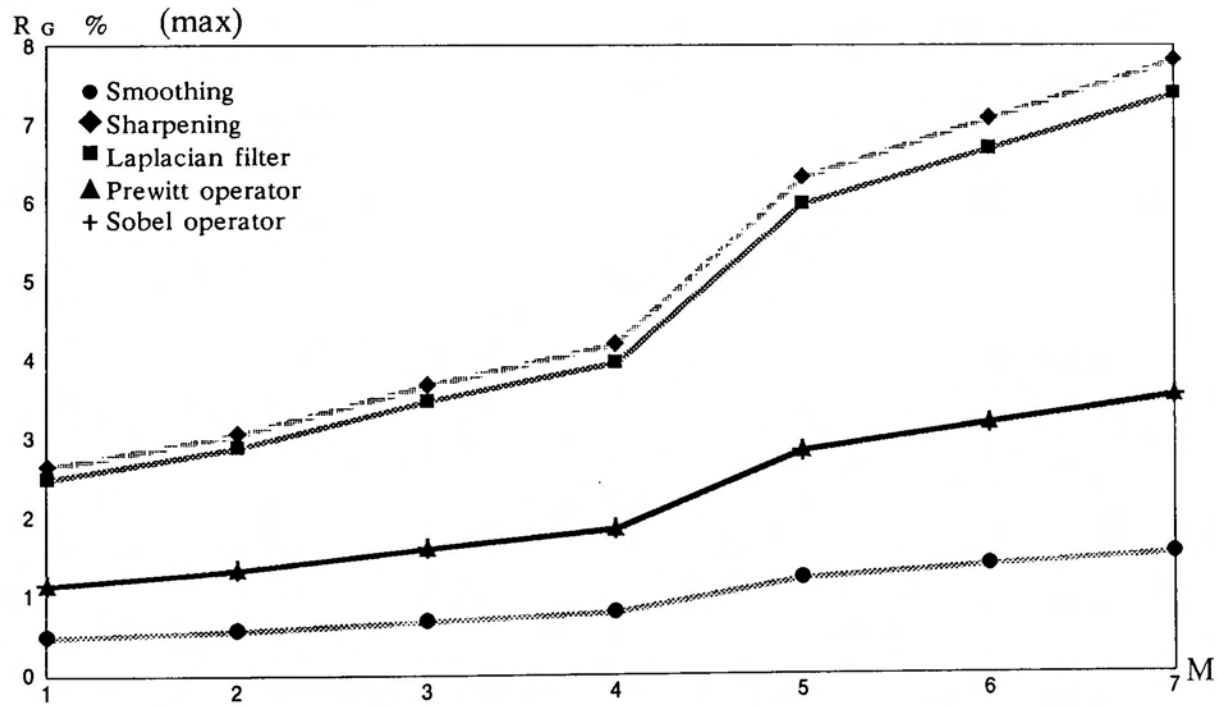

Fig.6.25. Relationship between experimental global memory accessing rate and the number of processors.

Table 6.25
Degree of Speed Up in Each Case

Processing Type	M=2	M=3	M=4	M=5	M=6	M=7
Gray-level Histogram	1.98	2.88	3.77	4.51	5.39	6.23
Differential Histogram	2.00	2.84	3.99	4.96	5.64	6.64
Sharpening	1.99	3.00	3.94	4.81	5.78	6.77
Smoothing	2.00	3.00	3.99	4.96	6.00	7.00
Laplacian Filtering	1.99	3.00	3.94	4.82	5.80	6.78
Sobel Operator	1.92	2.90	3.81	4.72	5.70	6.70
Prewitt Operator	2.00	3.00	3.97	4.91	5.94	6.97
Kirsch Operator	2.00	3.00	3.99	4.96	6.00	7.00

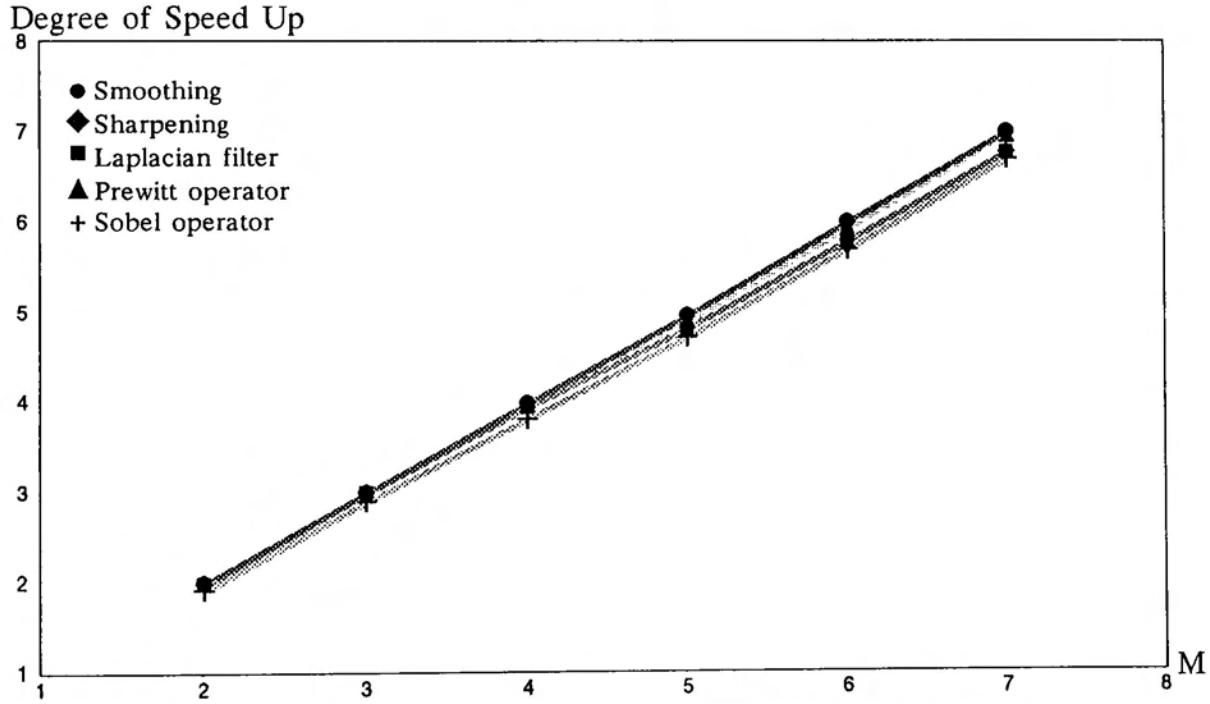

Fig.6.26. Degree of speed up with the increment of the number of processor.

Chapter 7

CONCLUSIONS AND DISCUSSIONS

7.1 Conclusions

The architecture of HIGIPS is the combination of pipeline processor architecture and multiprocessor architecture, i.e., the whole HIGIPS works at pipeline mode, and each stage of HIGIPS works at multiprocessor mode. This kind of architecture is suitable for the real-time image processing and analysis because all PE's in each stage can work in parallel at pixel level and all stages can work in parallel at image frame level. HIGIPS can work as a SIMD machine by downloading the same program to all PE's of every stage and as a MIMD machine by downloading different programs to the different stages.

The common bus speed is 842 kilobytes per second. It can transfer 6 image frames with the size of 200×640 bytes, 13 image frames with the size of 200×320 bytes, 26 image frames with the size of 200×160 bytes which is near to the video-rate (30 frame/s). So HIGIPS can handle the real-time image processing. But if to handle the high resolution images, the transfer rate (frames per second) will decrease.

HIGIPS can also handle the full color image data. But its data size is 3 times of that of black-white image data.

The traditional system AAP[48] (with 256×256 PE's) takes 25 seconds to perform the calculation of histogram for the input image with the size of 6800×4800 bytes. SIPS [36, 116] (with 87 PE's) performs the edge detection at 30 frames per second for the input image with the size of 768×512 bytes. And IDATEN[47, 75, 76] (with maximum 16 PE's) can performs the edge detection, moving object extraction at 30 frames per second from the input image with size of 512×512 bytes.

However, processing time can be reduced from $\tau_{in}+\tau_{out}+\sum\tau_i$ to maximum of τ_i, τ_{in}, and τ_{out} $(i=1, 2, \ldots, N)$ by using HIGIPS in comparison with the individual processor. Where τ_{in} is image input time of PIU, τ_{out} is final result output time of POU, and τ_i is processing time of PPU_i $(i=1, 2, \ldots, N)$. Each stage of prototype HIGIPS employs the common bus type multiprocessor. The bottle-neck problem in this kind of system can be avoided by the software techniques.

The PE of HIGIPS is the general-purpose microprocessor (μPD70126) which has software compatiblility with that of NEC PC-9801 series personal computer. HIGIPS does not

need any new programming environment. The monitor program of HIGIPS was developed with C language (MS–C, except a start up routine is written by assembler language). At present, the high-level language C of NEC PC-9801 personal computer is completely applicable to it. This provides the user with a great flexibility to develop the parallel programming for image processing and analysis.

From the experimental results, it is clear that HIGIPS is a high-speed and high-performance machine for image processing and analysis.

7.2 Discussions

HIGIPS is designed and developed for both low-level and high-level image processing and understanding. Its architecture is the combination of the multiprocessor and pipeline architecture to obtain high performance. But this does not mean that the more the stages of the HIGIPS, and the more the processors in one stage, the higher the performance of HIGIPS is. If the number of stages is large enough, pipeline travel time becomes an important factor unable to be neglected, and if the number of processors in one stage is large enough, the image segmentation becomes complicated and the data moving from GM into LM of each PE in a stage will take much more time than image processing. Thereby, there should exist a optimal combination of the number of stages and that of the processors in each stage in order to achieve the best performance of the system.

The common bus transfer rate may limit the processing speed of the whole system at some extent. If to desire a higher bus transfer speed than 842 kilobytes per second, a new high-speed common bus mechanism must be developed. This is a very important parameter to the new version of HIGIPS.

The HIGIPS is a parallel machine specialized for image processing and analysis. But it is a flexible general-purpose parallel machine in this field by downloading the different programs corresponding to applications. The HIGIPS can be used for studying and developing the new parallel algorithms of the high-level image processing and recognition, and the new non-Neuman type computer architecture.

Because the pipeline performance of HIGIPS is based on its memory module construction, it can be divided into multiple pipelines, e.g., bi-directional pipeline, this makes the HIGIPS perform the image processing flexibly, and also makes it be able to be applied to other fields, e.g., the research of neural network, neurophysiology, artificial intelligence.

All of these are left to be done in the future.

BIBLIOGRAPHY

[1] J. A. Abraham, P. Banerjee, C. Y. Chen et al, "Fault Tolerance Techniques for Systolic Arrays". *Computer,* July 1987, pp.65–87.

[2] G. A. Anderson and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples". *Computing Surveys,* Vol. 7, No. 4, December 1975, pp.197-213.

[3] R. J. Anderson and L. Snyder, "A Comparison of Shared and Nonshared Memory Models of Parallel Computation". *Proceedings of the IEEE,* Vol. 79, No. 4, April 1991, pp.480-487.

[4] M. Annaratone, E. Arnould, T. Gross et al, "The Warp Computer: Architecture, Implementation, and Performance". *IEEE Transactions on Computers.* Vol. C–36, No. 12, December 1987, pp.1523–1538.

[5] W. C. Athas and C. L, Seitz, "Multicomputers: Message–Passing Concurrent Computers". *Computer*, August 1988, pp.9–24.

[6] D. M. Balston, "A High-level Language for Constructing Image Processing Commands". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.99-116.

[7] J. L. Basille, S. Castan and J. Y. Latil, "Systeme Multiprocesseur Adapte au Traitement d'Images". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.205-213.

[8] K. E. Batcher, "Design of A Massively Parallel Processor". *IEEE Transactions on Computers*, Vol. C-29, No. 9, September 1980, pp.836-840.

[9] K. E. Batcher, "Bit-Serial Parallel Processing Systems". *IEEE Transactions on Computers*, Vol. C-31, No. 5, May 1988, pp.377-384.

[10] P. A. Bernstein, "Sequoia: A Fault–Tolerant Tightly Coupled Multiprocessor for Transaction Processing". *Computer*, February 1988, pp.37–45.

[11] W. Black, J. F. Harris and T. Clement, "A Generalized Support Package for Image Acquisition". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.117-123.

[12] W. J. Bouknight, S. A. Denenberg et al, "The Illiac System". *Proceedings of the IEEE*, Vol. 60, No. 4, April 1972, pp.369-388.

[13] F. A. Briggs, K. S. Fu et al, "PUMPS Architecture for Pattern Analysis and Image Database Management", *IEEE Transactions on Computer*, Vol. C-31, No. 10, October 1982, pp.969-983.

[14] V. Cantoni and S. Levialdi, "Multiprocessor Computing for Images". *Proceedings of the IEEE*, Vol. 76, No. 8, August 1988, pp.959-969.

[15] V. Cantoni, M. Ferretti and L. Lombardi, "A Comparison of Homogeneous Hierarchical Interconnection Structures". *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.416-428.

[16] P. E. Danielsson and S. Leviadi, "Computer Architectures for Pictorial Information Systems". *Computr*, November 1981, pp.53-67.

[17] T. Diede, C. F. Hagenmaier et al, "The Titan Graphics Supercomputer Architecture". Computer, September 1988, pp.13–30.

[18] R. J. Douglass, "MAC: A Programming Language for Asynchronous Image pProcessing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.41-51.

[19] B. L. Drake, F. T. Luk, J. M. Speiser et al, "SLAPP: A Systolic Linear Algebra Parallel Processor". *Computer*, July 1987, pp.45–49.

[20] M. Dubois and C. Scheurich, "Synchronization, Coherence, and Event Ordering in Multiprocessors". *Computer*, February 1988, pp.9–21.

[21] P. M. Flanders, R. L. Hellier et al, "Efficient High-level Programming on the AMT DAP". *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.524-536.

[22] S. A. Felperin, L. Gravano et al, "Routing Techniques for Massively Parallel Communication". *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.488-503.

[23] M. J. Flynn, "Some Computer Organizations and Their Effectiveness". *IEEE Transactions on Computers*, Vol. C-21, No. 9, September 1972, pp.948-960.

[24] M. J. Flynn, "Very High-speed Computing Systems". *Proceedings of the IEEE*, Vol. 54, No. 12, December 1966, pp.1901-1909.

[25] J. A. B. Fortes, "Systolic Arrays —From Concept to Implementation". *Computer*, July, 1987, pp.12–17.

[26] J. A. B. Fortes, "Systolic Arrays: A Survey of Seven Projects". *Computer*, July, 1987, pp.91–103.

[27] D. E. Foulser, "The Saxpy Matrix–1: A General–Purpose Systolic Computer". *Computer*, July 1987, pp.35–43.

[28] T. J. Fountaint, "CLIP4: A Progress Report". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.283-291.

[29] Y. Fujita, T. Ishiguro, M. Yamazaki et al, "A Data Flow Image Processing System TIP–4 Prototype". *Technical Report of Information Processing Society of Japan*, Vol.88 No.73, CV56 (in Japanese).

[30] B. Furht, V. Milutinovic, "A Survey of Microprocessor Architectures for Memory Management". *Computer*, March 1987, pp.48–67.

[31] P. Gemmar, H. Ischen and K. Luetjen, "FLIP: A Multiprocessor System for Image Processing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.245-256.

[32] F. A. Gerritsen and R. D. Monhemius, "Evaluation of the Delft Image Processor DIP-1". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.189-203.

[33] F. A. Gerritsen and L. G. Aardema, "Design and Use of DIP-1: A Fast, Flexible and Dynamically Microprogrammable Pipelined Image Processor". *Pattern Recognition*, Vol. 14, Nos. 1-6, pp.319-330.

[34] G. H. Granlund, "GOP: A Fast and Flexible Processor for Image Analysis". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.179-188.

[35] B. Gudmundsson, "Overview of the High-level Language for PICAP". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.147-156.

[36] A. Hasebe, R. Kato, N. Ito and A. Kikuchi, "SIPS: A Multiprocessor System for Video Image Processing". *Technical report of Information Processing Society of Japan*, Vol. CV-39-5, November 21,1985 (in Japanese).

[37] D. J. Hunt, "The ICL DAP and its Application to Image Processing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.275-282.

[38] T. Isonishi, H. Miyata and A. Iwase, "An Architecture of Cellular Array Processor for Image Processing". *Technical report of Information Processing Society of Japan*, Vol.88 No.79, ARC73-9, pp.61-68 (in Japanese).

[39] K. E. Jordan, "Performance Comparison of Large-Scale Scientific Computers: Scalar Mainframes, Mainframes with Integrated Vector Facilities, and Supercomputers". *Computer*, March 1987, pp.10-23.

[40] K. Kani, Ed., "*V Series Microcomputer I*", Maruzen Inc., December 1985 (in Japanese).

[41] A. Kasai, Ed., "*Practical Macroassembler ver.3.0/4.0*", Gijyutsu Hyouronn Sya, June, 1987 (in Japanese).

[42] M. Kidode and H. Shinoda, "A Survey of Special Hardware Architecture for High Speed Image Processing". In *Nikkei Electronics*, No. 191, July 1978, pp.110-140 (in Japanese).

[43] M. Kidode, H. Asada, H. Shinoda et al, "Local Parallel Processor". *Toshiba Review*, Vol. 34, No. 6, 1979, pp.511-514 (in Japanese).

[44] M. Kidode, M. Tabata and N. Aihara, "High-Speed Compact Image Processing system, TOSPIX". *Toshiba Review*, Vol. 37, No. 12, 1982, pp.1047-1050 (in Japanese).

[45] M. Kidode, "Image Processing Machines in Japan". *Computer*, Vol. 16-1, January 1983, pp.68-80.

[46] M. Kidode and K.Sakaue, "New trends of Special Hardware Architectures for Image Processing". *Nikkei Electronics*, No. 295, July, 1982, pp.179-212 (in Japanese).

[47] M. Komeichi, S. Sasaki, T. Ozaki et al, "A Color Time-varying Image Processing System: 'color-IDATEN'". *Technical Report of Information Processing Society of Japan*, Vol. CV-49-2, July 23, 1987 (in Japanese).

[48] T. Kondo, T. Nakashima, M. Aoki and T. Sudo, "An LSI Adaptive Array Processor". *IEEE JSSC*, Vol. SC-18, No. 2, April 1983, pp.147-156.

[49] Z. Kulpa, "PICASSO, PICASSO-SHOW and PAL—A Development of a High-level Software System for Image Processing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.13-24.

[50] S. Y. Kung, S. C. Lo, S. N. Jean and J. N. Hwang, "Wavefront Array Processors--Concept to Implementation". Computer, July 1987, pp.18–33.

[51] T. Kushner, A. Y. Wu and A. Rosenfeld, "Image Processing on ZMOB". *IEEE Transactions on Computers*, Vol. C-31, No. 10, October 1982, pp.943-951.

[52] G. R. Lang, M. Dharssi, F. M. Longstaff et al, "An Optimum Parallel Architecture for High–Speed Real–Time Digital Signal Processing". *Computer*, February 1988, pp.47–59.

[53] C. Lantuejoul, "An Image Analyzer". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.165-177.

[54] R. M. Lea and I. P. Jalowiecki, "Associative Massivly Parallel Computers". *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.469-479.

[55] S. Levialdi, A. Maggiolo-Schettini, M. Napoli, G. Tortora and G. Uccella, "On the Design and Implementation of PIXAL, A Language for Image Processing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.89-98.

[56] H. Li and Q. F. Stout, "Reconfigrable SIMD Massively Parallel Computers". *Proceedings of the IEEE*, Vol.79, N0. 4, April 1991. pp.429-443.

[57] D. J. Lilja, "Reducing the Branch Penalty in Pipelined Processors". *Computer*, July 1988, pp.47–55.

[58] K. Luetjen, P. Gemmar and H. Ischen, "FLIP: A Flexible Multiprocessor System for Image Processing". *Proceedings of 5th International Conference on Pattern Recognition*, Miami Beach, Florida, December 1-4, 1980, Vol. 1, pp.326-328.

[59] A. Maggiolo-Schettini, "Comparing some High-level Languages for Image Processing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.157-164.

[60] R. Manara and L. Stringa, "The EMMA System: An Industrial Experience on a Multiprocessor". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.215-227.

[61] M. Maresca, M. A. Lavin and Hungwen Li, "Parallel Architectures for Vision". *Proceedings of the IEEE*, Vol. 76, No. 8, August 1988, pp.970-981.

[62] Y. Masaki, K. Hori, H. Uchino and Y. Okano, "Real-time Image Processing System—RAPID—". *Technical report of Information Processing Society of Japan*, Vol. CV-50-2, October 1, 1987 (in Japanese).

[63] J. V. McCanny, J. G. McWhirter, "Some Systolic Array Developments in the United Kingdom". *Computer*, July 1987, pp.51–63.

[64] T. Mori, M. Maruyama, K. Aono et al, "Real-time Image Signal Processor RISP-II". *Technical Report of the Institute of Electronics and communication engineers.* Vol. 86 No. 191, SSD86–96(in Japanese).

[65] T. N. Mudge, J. P. Hayes, and D. C. Winsor, "Multiple Bus Architectures". *Computer*, June 1987, pp.43–48.

[66] J. J. Navarro, J. M. Llaberia, and M. Valero, "Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array Processors". *Computer*, July 1987, pp.77–89.

[67] K. M. Nichols and J. T. Edmark, "Modeling Multicomputer Systems with PARET". *Computer*, May 1988, pp.39–48.

[68] L. Norton-Wayne, "High-level Image Languages for Automatic Inspection—the File structure Problem". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.139-146.

[69] J. L. Potter, "Image Processing on the Massively Parallel Processor". *Computer*, Vol. 16-1, January 1983, pp.62-67.

[70] K. Preston, Jr, "Comparison of Parallel Processing Machines: A Proposal". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.305-319.

[71] T. Radhakrishnan, R. Barrera, A. Guzman and A. Jinich, "Design of a High-level Language (L) for Image Processing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.25-40.

[72] J. T. Rayfield and H. F. Silverman, "System and Application Software for the Armstrong Multiprocessor". *Computer*, June 1988, pp.38–52.

[73] D. A. Reed and D. C. Grunwald, "The Performance of Multicomputer Interconnection Networks". *Computer*, June 1987, pp.63–73.

[74] S. Sakai, Y. Yamaguchi et al, "Architecture of a Dataflow Single Chip Processor EMC-R". *Technical report of Information Processing Society of Japan*, Vol.88 No.45, ARC71-3, PP.17-24 (in Japanese).

[75] S. Sasaki, T. Satoh and M. Yoshida, "Reconfigurable Video-Rate Image Processing System: IDATEN". *FUJITSU*, Vol. 37, No. 3, pp.230-236, May 1986 (in Japanese).

[76] S. Sasaki, T. Satoh, H. Iwase and T. Goto, "IDATEN: A Real-time Image Processing system with Modifiable Network". *Technical report of Information Processing Society of Japan*, Vol. CV-37-1, July 18, 1985 (in Japanese).

[77] M. Sato, H. Matsuura et al, "Multimicroprocessor System PX-1 for Pattern Information Processing". *The Transactions of the Institute of Electronics, Information and Communication Engineers*, Vol. J64-D, No. 11, 1981, pp.1021-1029 (in Japanese).

[78] D. H. Schaefer, "The Characterization and Representation of Massively Parallel Computing Structures". *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.461-468.

[79] H. J. Siegel, "Interconnection Networks for SIMD Machines". *Computer*, June 1979, pp.57-65.

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

[81] H. J. Siegel, "PASM: A Reconfigurable Multimicrocomputer System for Image Processing". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.257-265.

[82] L. J. Siegel, "Image Processing on a Partitionable SIMD Machine". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.293-300.

[83] J. M. Sipelstein and G. E. Blelloch, "Collection-Oriented Languages". *Proceedings of the IEEE*, Vol. 79, No. 4, April, 1991, pp.504-523.

[84] D. B. Skillicorn, "A Taxonomy for Computer Architectures". *Computer*, November 1988, pp.46-57.

[85] J. A. Stankovic, "A Serious Problem for Next-Generation Systems'. *Computer*, October 1988, pp.10-19.

[86] P. Stenstrom, "Reducing Contention in Shared-Memory Multiprocessors". *Computer*, November 1988, pp.26-36.

[87] J. P. Strong, "Computations on Massively Parallel Processor at the Goddard Space Flight Center". *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.548-558.

[88] S. Sugimoto and Y. Ichioka, "High Speed Image Processing System With Time Shared Multiframe Data Bus: MFIP". *The Transactions of the Institute of Electronics, Information and Communication Engineers*, Vol. J68-D No. 4, 1985, pp.901-908.

[89] Y. Takahashi, "Processor Interconnection Systems for Parallel Processors". *Information Processing*, Vol. 23 No. 3, 1982, pp.201-209 (in Japanese).

[90] Y. Takaki, S. Horiguchi et al, "Software Architecture for the Multiprocessor System MUGEN", *Technical Report of Information Processing Society of Japan*, Vol.87 No.78, CA68-6, pp.41-48 (in Japanese).

[91] K. Tanabe, Ed., "*8086 Microcomputer*", Maruzen Inc., March 1983 (in Japanese).

[92] S. L. Tanimoto, "*Memory Systems for Highly Parallel Computers*", *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.403-415.

[93] T. Temma, M. Mizoguchi and S. Hanaki, "Image Processor by flexible Pipeline Approach: TIP-1". *The Transactions of the Institute of Electronics, Information and Communication Engineers*, Vol. J68-D No. 4, April, 1985, pp.853-860 (in Japanese).

[94] T. Temma, M. Mizoguchi and S. Hanaki, "The Architecture and Data-flow Simulation of Template-controlled Image Processor (TIP)". *Technical Report of the Institute of Electronics and Communication Engineers*, IE81-6, pp.41-48 (in Japanese).

[95] L. W. Tucker and G. G. Robertson, "Architecture and Applications of the Connection Machine". *Computer*, August 1988, pp.26-38.

[96] M. M. Trivedi and A. Rosenfeld, "On Making Computers "See"". *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 19, No. 6, November/December 1989, pp.1333-1335.

[97] L. Uhr, "A Language for Parallel Processing of Arrays, Embedded in Pascal". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.53-87.

[98] J. Vrolijk, P. L. Pearson and J. S. Ploem, "TAL: An Interpretive Language for the Leyden Television Analysis System". In *Languages and Architectures for Image Processing*, M. J. B. Duff, S. Levialdi, Eds, New York, NY: Academic Press, 1981, pp.125-137.

[99] D. L. Waltz, "Applications of the Connection Machine". *Computer*, January 1987, pp.85–97.

[100] C. C. Weems, "Architectural Requirements of Image Understanding with Respect to Parallel Processing". *Proceedings of the IEEE*, Vol. 79, No. 4, April 1991, pp.537-547.

[101] P. Wolcott and S. E. Goodman, "High–Speed Computers of the Soviet Union". *Computer*, September 1988, pp.32–41.

[102] N. Yagi, R. Yajima, K. Enami et al, "Picot-system—Real-time Video Signal Processing System". *Technical report of the Institute of Electronics, Information, and Communication Engineers*, Vol. PRU88-100, 1988, pp.3-10 (in Japanese).

[103] N. Yagi, R. Yajima, K. Enami et al, "Real-time Video Signal Processing LSI—Picot". *Technical report of the Institute of Electronics, Information, and Communication Engineers*, Vol. ICD88-35, 1988, pp.63-70 (in Japanese).

[104] F. H. Yao, A. Tamaki and K. Kato, "The Development of a High-speed General-purpose Image Processing System with Microprocessors". *Record of 1987 Joint Conference of Electrical and Electronics Engineers in Kyushu*, pp.415 (in Japanese).

[105] F. H. Yao, A. Tamaki and K. Kato, "A Study of the Architecture of High-speed General-purpose Image Processing System—HIGIPS". *Proceedings of National Conference of Information Processing Society of Japan*, 1988, pp.1477-1478 (in Japanese).

[106] F. H. Yao, A. Tamaki and K. Kato, "The Software Development of the High-speed General-purpose Image Processing System—HIGIPS: In the Case of Arm Movement Recognition". *Record of 1988 Joint Conference of Electrical and Electronics Engineers in Kyushu*, pp.639 (in Japanese).

[107] F. H. Yao, A. Tamaki and K. Kato, "Monitor of High-speed General-purpose Image Processing System—HIGIPS: Facilities of the Monitor between SUB-controller and Slave MPU". *Proceedings of National Conference of Information Processing Society of Japan*, 1989, pp.1026-1027 (in Japanese).

[108] F. H. Yao, A. Tamaki and K. Kato, "Monitor of High-speed General-purpose Image Processing System—HIGIPS: Facilities of the Monitor between SC and SUBC". *Record of 1989 Joint Conference of Electrical and Electronics Engineers in Kyushu*, pp.541 (in Japanese).

[109] F. H. Yao, A. Tamaki and K. Kato, "The Development of Image Input/Output Units of High-speed General-purpose Image Processing System—HIGIPS". *Record of 1990 Joint Conference of Electrical and Electronics Engineers in Kyushu*, pp.611 (in Japanese).

[110] F. H. Yao, A. Tamaki and K. Kato, "The Development of HIGIPS—the High-speed General-purpose Image Processing System". *Proceedings of International Computer Symposium 1990,* Taiwan, Dec.17-19, 3E-4.

[111] F. H. Yao, A. Tamaki and K. Kato, "A High-speed General-purpose Image Processing System HIGIPS". To appear in: *The Transactions of the Institute of Electronics, Information and Communication Engineers* (in Japanese).

[112] F. H. Yao, A. Tamaki and K. Kato, "Image Processing on HIGIPS". To appear in: *Proceedings of* **AMSE** *International Confference on Infromation and Systems '91.* Hangzhou, China, October 9-11, 1991.

[113] F. H. Yao, A. Tamaki and K. Kato, "An Aplication of HIGIPS to Recognition of Pendulum Like Motion". To appear in: *Proceedings of International Confference on Industrial Electronics, Control and Instrumentation '91.* Kobe, Japan, October 28-November 1, 1991.

[114] F. H. Yao, A. Tamaki and K. Kato, "Recognition of 3-D Motion of a Human Arm with HIGIPS". To appear in: *Proceedings of '91 Korea Automatic Control Conference.* Seoul, Koea, October 22-24, 1991.

[115] F. H. Yao, A. Tamaki and K. Kato, "HIGIPS Machine: A High-speed General-purpose Image Processing System". To appear in: *Proceedings of ISMM International Symposium '91.* Long Beach, California, USA, December 16-18, 1991.

[116] H. Yoshida and A. Hasebe, Eds. "Real-time Image Processing". Nikkei Magurou-Sya, 1986 (in Japanese).

[117] *"CUPL 2.1: The Universal compiler for Programmable Logic, User's Manual"*, Personal CAD Inc..

[118] *"MS-DOS Macroassembler User's and Reference Manual"*, NEC, June,1988 (in Japanese).

[119] "NEC: μPD70216/G/L/R-8 Data Sheet". In *DATUM*, November 1986, pp.135-268 (in Japanese).

[120] "Special Issue: Details about PC9801 and its Extended Interfaces". In *Transistor Technology, Special*, No.3, CQ-Shuppansya (in Japanese).

[121] "Special Issue: Thorough Study on V40/V50". In *Transistor Technology*, April 1988, pp.343-420 (in Japanese).

[122] "Special Issue: Study the Programming Techniques from Practice". In *Transistor Technology*, January 1989, pp.411-492 (in Japanese).

[123] "HyPER VISION: User's Manual". Digital Arts Ltd..

[124] "HyPER FRAME: User's Manual". Digital Arts Ltd..

[125] "8288, 8289 Data Sheets". Intel Component Data Catalog, pp.8-126 - 8-143.

VITA

Feng-Hui Yao was born in Hebei province, the People's Republic of China, in 1963. He received his B.S. degree in Electronic Communication from Dalian Marine University, Dalian, Liaoning province, in July 1984. In October 1985, he was awarded a scholarship from the Ministry of Education of Japanese government to pursue his M.S. in Computer Science at Kyushu Institute of Technology, Kitakyushu, and obtained his M.S. degree in March 1988. His research interests are image processing and computer vision, computer architecture, parallel processing.