Highly-parallel Two-dimensional Cellular Automaton Architecture and its Application to Real-time Image Processing

Takeshi Ikenaga

March 2001

Summary

The focus of this dissertation is the design of a highly-parallel computer for real-time image-understanding processing, one that is extremely compact and provides very high levels of performance. This computer can be used for implementation of various image-understanding applications in the fields of industrial inspection, medical imaging, intelligent transportation systems (ITS), robotics, multimedia, human interface, entertainment, image coding, and so forth.

The result is CAM^2, which stands for Cellular AutoMata on Content-Addressable Memory. As the name shows, this new architecture combines cellular automaton (CA) and content-addressable memory (CAM). A CA is a promising computer paradigm that can break through the von Neumann bottleneck. A two-dimensional CA is especially suitable for application to pixel-level image processing because an image has a two-dimensional topology. Although various architectures for processing two-dimensional CA have been proposed, there are no compact, practical computers. So, in spite of its great potential, CA is not widely used. To remedy this situation, I can think of a CAM turned to CA processing. CAM, or an associative processor, that can perform various types of parallel processing with words as the basic unit is a promising component for creating a compact CA cell array because of its suitability for LSI implementation. CAM makes it possible to embed an enormous number of processing elements (PEs), corresponding to CA cells, in one VLSI chip. It can attain pixel-order parallelism on a single board.

There are three basic problem areas that must be integrated in order to produce a high-performance, compact, and flexible CAM^2 and demonstrate its usefulness for real-time image-understanding processing. These three areas are

1. computer architecture,

2. LSI and system design and implementation, and

3. applications.

In the computer architecture area, the study effort is focused on the following three architectural considerations: CA mapping, CA processing, and data loading and retrieval processing. Multiple-zigzag mapping enables two-dimensional CA cells to be mapped into CAM words, even though physically a CAM has a one-dimensional structure. Dedicated CAM functions enable high-performance CA processing. Furthermore, parallel loading and partial word retrieval techniques enable high-throughput data transfer between CAM^2 and the outside image buffer. The performance evaluation results show that 256 k CA cells, which correspond to a 512 $\times$ 512 pixel picture, can be processed by a CAM^2 on a single board using deep sub-micron process technology. Moreover, the processing speed is more than 10 billion CA cell updates per second (CUPS). This means that more than a thousand CA-based image processing operations can be performed on a 512 $\times$ 512 pixel image at video rates (33 msec). These results demonstrate that CAM^2 will represent a major step toward the development of a compact and high-performance two-dimensional cellular automaton.

In the design and implementation area, fully-parallel 1-Mb CAM LSIs with dedicated functions for CA processing have been designed and fabricated, as has a proof-of-concept prototype CAM^2 system (PC board) using these LSIs. To satisfy the extremely severe design constraints of the state-of-the-art process technology (0.25 μm), this study involves not only VLSI circuit design, but also packaging technology, circuit board fabrication technology, power and signal distribution techniques, heat dissipation problems and design and verification strategy. The CAM chip capable of operating at 56 MHz with 2.5-V power supply was fabricated using 0.25-μm full-custom CMOS technology with five aluminum layers. A total of 15.5 million transistors have been integrated into a 16.1$\times$17.0-mm chip. Typical power dissipation is 0.25 W. The processing of various update and data

transfer operations was performed at 3-640 GOPS. The fabricated CAM has 16 k words, or processing elements (PEs), which can process 128 $\times$ 128 pixels in parallel, and a board-sized pixel-parallel image processing system can be implemented using several chips. Indeed, the prototype board has a two-dimensional array (2 $\times$ 2) of CAM chips, and can handle a 256 $\times$ 256 pixel image. Since PCI bus and NTSC video interfaces are also embedded in the board, a compact image-processing platform can be built simply by connecting the board to a personal computer and a video camera. The LSI and the board confirm that an economically feasible, compact, high-performance, and flexible CAM^2 can be actually obtained with the current technology.

In the application area, two rather advanced computation paradigms based on CA are taken up: discrete-time cellular neural network (DTCNN) and mathematical morphology. DTCNN is a promising computer paradigm that fuses artificial neural networks (ANN) with the concept of CA. Mathematical morphology is an image transformation technique that locally modifies geometric features through set operations. Both are becoming powerful tools with various applications in image processing field. Here, efficient mapping and processing methods to perform various kinds of DTCNN and morphology processing are studied. Evaluation results show that, on average, CAM^2 can perform one transition for various DTCNN templates in about 12 μsec. CAM^2 also can perform one morphological operation for basic structuring elements within 30 μsec. These results mean that more than a thousand operations can be carried out on an entire 512 $\times$ 512 pixel image at video rates (33 msec). Furthermore, CAM^2 can perform practical image processing through a combination of DTCNN, morphology, and other algorithms. These results demonstrate that CAM^2 will enable fuller realization of the potential of DTCNN and morphology and contribute significantly to the development of real-time image processing systems based on DTCNN, morphology and a combination of them.

Acknowledgements

First and foremost, I express my sincere appreciation to Professor Katsuhiko Shirai of Waseda University for his encouragement, constant guidance, and support during the writing of this dissertation. I also express my gratitude to Professor Yoichi Muraoka, Professor Masao Yanagisawa and Professor Hayato Yamana of Waseda University for reviewing this work. I also thank Professor Shigeki Goto and the other faculty members at Information & Computer Science Department of Waseda University for their insightful comments at the public hearing.

The research reported in this dissertation was conducted from 1996 to 1999 in NTT LSI Laboratories, NTT System Electronics Laboratories, NTT Integrated Information & Energy Systems Laboratories and NTT Lifestyle and Environmental Technology Laboratories. I express my deep thanks to Mr. Yasuyoshi Sakai (Advanced Telecommunications Research Institute International (ATR)), Mr. Masataka Hirai (NTT AFTY Co.) Dr. Ken-ichi Nakano (NTT Advanced Technology Co.), Dr. Ichiro Yamada (NTT Lifestyle and Environmental Technology Labs.), Dr. Osamu Karatsu (SRI consulting K. K.), Mr. Ken Takeya (NTT Telecommunications Energy Labs.), Dr. Ryota Kasai (NTT Electronics Co.), Dr. Shoji Makino (NTT Communication Science Labs.), and Dr. Hitoshi Kitazawa (NTT Lifestyle and Environmental Technology Labs.) for their administrative support and encouragement.

I express my sincere thanks to my two mentors on CAM-based computer vision hardware: Dr. Takeshi Ogura (NTT Lifestyle and Environmental Technology Labs.) and Professor Charles C. Weems (University of Massachusetts, Amherst) for their direct guidance, criticism, helpful suggestions, and continuous encouragement during the research. I

also thank Professor Hironori Yamauchi (Ritsumeikan University) and Dr. Jun-ichi Takahashi (Deloitte Tohmatsu Consulting) for giving me the opportunity to start research in the LSI and computer architecture fields and for their guidance.

Furthermore, I thank Professor Toshio Kondo (Mie University), Professor Shinji Yamamoto (Toyohasi University of technology), Professor Keikichi Tamaru, Professor Hidetoshi Onodera, Dr. Kazutoshi Kobayashi (Kyoto University), Dr. Katsunori Shimohara, Mr. Hitoshi Henmi (Advanced Telecommunications Research Institute International (ATR)), Mr. Kuniharu Uchimura, Mr. Hisao Nojima, Mr. Mamoru Nakanishi, Ms. Namiko Ikeda, Ms. Miki Kitabata (NTT Lifestyle and Environmental Technology Labs.), Mr. Kimihisa Aihara (NTT Science and Core Technology Laboratory Group), Mr. Yuichi Fujino, Mr. Tadashi Nakanishi (NTT Cyber Solutions Labs.), Mr. Makoto Endo, Dr. Jiro Naganuma, Mr. Takeshi Yoshitome, Mr. Yutaka Tashiro, Mr. Hiroaki Matsuda, Mr. Koyo Nitta, Dr. Sakuichi Otsuka, Ms. Yasuko Takahashi (NTT Cyber Space Labs.), Mr. Junji Yamato (NTT Communication Science Labs.), Dr. Toshiaki Miyazaki, Dr. Shin-ichi Minato, Mr. Tsunemasa Hayashi(NTT Network Innovation Labs.), Mr. Tomoji Toriyama, Dr. Masahito Kawamori (NTT Service Integration Labs.), Mr. Osamu Saito (NTT Information Sharing Platform Labs.), Mr. Toshihiro Minami (NTT Intellectual Property Center), Mr. Ritsu Kusaba, Mr. Eiichi Hosoya (NTT Communications Co.), Mr. Tetsuya Abe, Mr. Kazuhito Suguri (NTT East), Mr. Tsuneo Okubo, Mr. Toshio Tsuchiya and Mr. Mitsuo Ikeda (NTT Electronics Co.) for their technical support and fruitful discussions, and Mr. Tatsuo Baba and Mr. Yasuharu Nakabayashi (NTT Electronics Co.) for contributions to the development of the CAM LSI.

Finally, I thank my parents, my wife, and my daughter and son for their kind support over the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and purpose of this dissertation

1.1.1 Cellular automaton as a computation paradigm for image processing

Von Neumann made two significant contributions to the history of digital computers. One was single central processing unit (CPU) computer architecture, which is generally called the von Neumann architecture. It is regarded as the most fundamental technology for creating stored-program sequential computers and is widely used to date. The other was his work in parallel computing via his research on arrays of computers. This array is called a cellular automaton (CA) [1], [2], and was originally conceived by von Neumann more than a half a century ago.

A CA is a dynamical system in which space and time are discrete. The CA consists of an array of identical computers or processing elements (PE) each with its own value and connected to its immediate neighbors. Figure 1.1 shows an example of a two-dimensional CA. CA processing is carried out by the iterative operations of CA-value transfer and update. A transfer operation involves transferring the values of neighboring cells to the original cell. An update involves calculating the next value for the original cell simultaneously in discrete steps by applying a local, identical interaction rule to the original and neighboring cell values. From the point of view of parallel machines, a CA exhibits three notable features: large-scale parallelism, local cellular interactions and simple basic components (cells). It can also be classified as a typical single instruction multiple data stream

(SIMD) model based on Flynn's taxonomy [3], [4] of parallel computers. Moreover, from the programming point of view, a CA offers an extremely simple environment, compared with other parallel processing models. Even though the mechanism is very simple, a CA is a promising computer paradigm that can break through the von Neumann bottleneck that greatly degrades the performance of high performance computers required to handle large volumes of data.

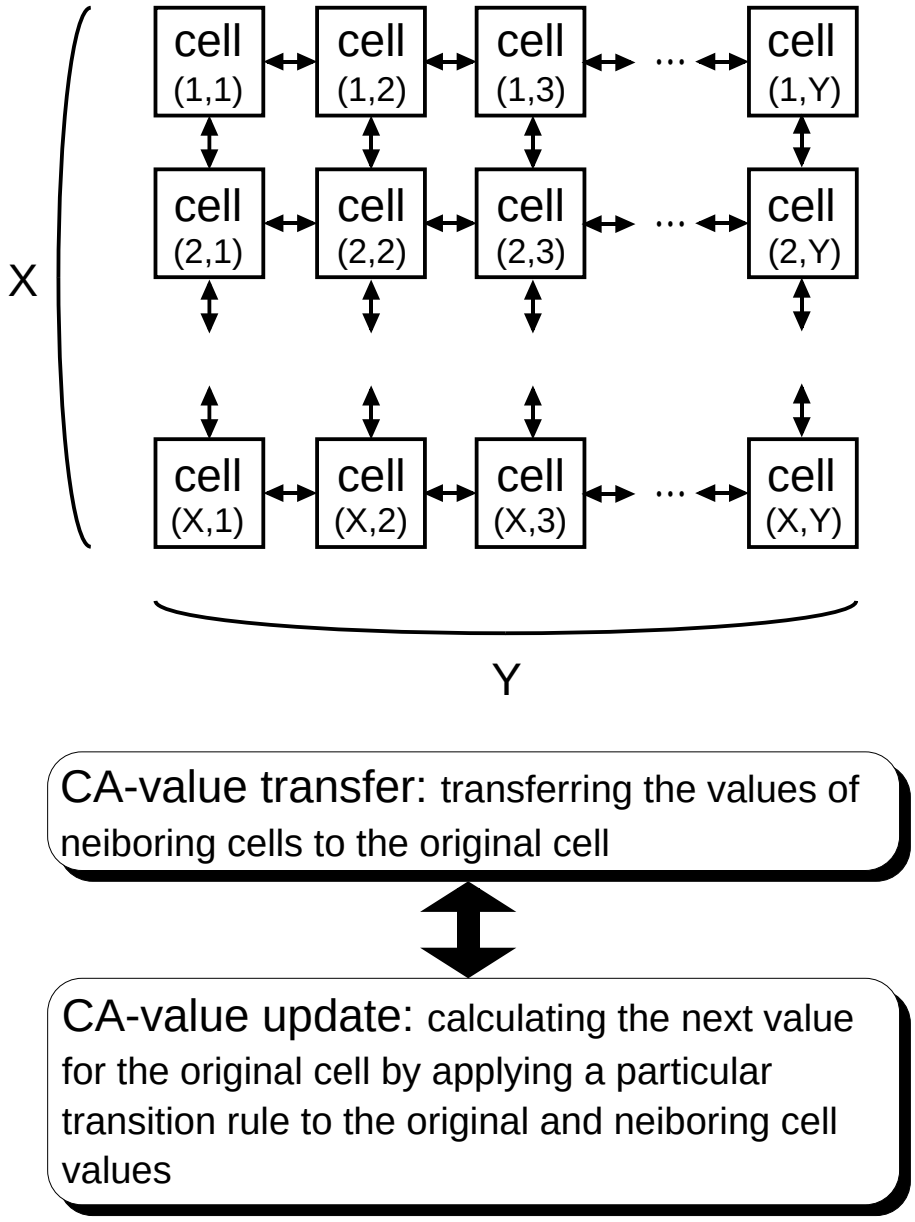


Figure 1.1: Model of two-dimensional cellular automaton.

Neumann's first objective with the CA was to find a precise way of dealing with the problem of how to have machines reproduce themselves [2]. The CA was considered to provide a formal framework for investigating the behavior of complex, extended systems. Various kinds of artificial life research have been undertaken based on this concept [5]. However, the CA offers far more possibility than those for which it was first intended; it is also a promising computation paradigm for other fields. In particular, a two-dimensional CA has many applications as regards pixel-parallel image processing and low (processing from image to image) to intermediate (processing from image to symbol) level vision processing [6], because an image has a two-dimensional topology. Indeed, most conventional pixel-parallel image processing algorithms, such as segmentation, skeletonization [2] and linear filtering [7], [8], can be easily mapped on a CA. Two rather advanced applications are discrete-time cellular neural networks (DTCNN) [9], [10] and mathematical morphology [11], [12].

In spite of its great potential and long history, a CA is not widely used in the field of image-understanding processing, probably because there are no compact, practical computers that can process real-world images of several hundred thousand pixels at video rates.

1.1.2 Architecture for 2D cellular automata

Vision plays an important role in the exchange of various kinds of information or even emotions in a wide range of human communication situations. Vison is also useful for obtaining various kinds of information from the real world. Thus, it is easy to believe that a vision system would have many applications both as a communicating medium and a rich sensor. Indeed, image-understanding processing is becoming indispensable for the implementation of various applications in the fields of industrial inspection, medical imaging, intelligent transportation systems (ITS), robotics, multimedia, human interface, entertainment, and image coding [8].

There are three factors that must be considered for a vision system designed to cover

this wide range of applications. The first factor is processing performance. Pixel parallel image processing and low to intermediate vision processing are indispensable for the above applications. However, this type of processing requires us to execute a very large number of logical and arithmetical operations for high-volume image data, especially for real-time applications, because it requires iterative update and data transfer operations for all pixel data. For example, performing a single operation on each pixel translates to 23 million instructions per second for a 512×512 pixel color image. In many cases, such operations must be repeated a large number of time and many vision researchers believe that one hundred thousand times the above number of operations is required [13]. It is almost impossible for general-purpose microprocessors to handle such processing no matter how fast they might become with the progress of LSI technology. Therefore, hardware with extremely high levels of performance and high-frequency memory access is highly desirable.

The second factor is system size. Since in most vision applications, a vision system is placed in front of a user (e.g. a PC or an information terminal for human interface applications) or is embedded in some kind of on-site box (e.g. a sensor box in ITS applications), the system should be compact; otherwise, it would not be practical to use it outside a laboratory.

The last factor is flexibility. Most vision algorithms consist of various kinds of operations and their combinations. Moreover, since the computer vision field is still a rapidly evolving area of research, new algorithms are constantly being developed and tested. Innovative algorithms may continue to appear. Thus, to cover this widering range of applications, the vision system should be as flexible as possible. This means that high levels of performance, compactness and flexiblity are key factors when designing a two-dimensional CA for image understanding.

Next, I survey conventional CA architectures for image processing and discuss their problems in terms of the above key factors. Conventional architecture falls into two categories: 2D SIMD cellular array and multiple-stage pipeline architecture.

2D SIMD cellular array architecture

A fully-parallel two-dimensional SIMD cellular array architecture has been proposed with a view to realizing a computer for CA, or a cellular logic computer (CLC) [14]. Many machines such as CLIP-4 [15], MPP [16], AAP-2 [17], CM-2 [18] and MP-1 [19] have been developed. The fully-parallel type, which consists of two-dimensional processing elements (PEs) and interconnection networks, is the most natural architecture for two-dimensional CA. The drawback of the conventional fully-parallel approach is the huge amount of hardware it requires. At most, only several dozen PEs can be embedded in one VLSI chip. Therefore, enormous numbers of VLSI chips are required if we wish to realize pixel-order (several hundred thousand) parallelism, which is crucial as regards CA performance. Moreover, two-dimensional interconnection networks cause I/O bottlenecks, so it is difficult to increase the number of PEs even if we adopt state-of-the-art LSI technology.

Multiple-stage pipeline architecture

Multiple-stage pipeline architectures, such as Cytocomputer [20] and CAM8 [21] have also been proposed for CLC. Since these machines are not true CA, they are also called cellular automaton machines. This architecture can achieve high-performance CA processing, but the hardware structure must be fully tuned to the application because the most suitable configuration is different for different applications. Due to this lack of flexibility, it is not truly suitable for practical pixel-level image processing that requires various CA processes.

Against this backdrop, it is very clear that conventional architectures are unsatisfactory solutions to the problem of developing compact, practical, and flexible CA for image-understanding processing. The purpose of this study is to develop a compact, high-performance, flexible highly-parallel two-dimensional cellular automaton for real-time image-understanding processing. The result is CAM^2, which stands for Cellular AutoMata on Content-Addressable Memory.

1.2 Key technologies: HiPIC and CAM

I focused on two important technical isues with a view to achieving the goal of my research, namely the development of a compact, high-performance, and flexible CA called CAM^2. These issues were the system model and the LSI architecture. The former is the key to creating a flexible system. The latter is the key to achieving both high levels of performance and compactness simultaneously.

1.2.1 System model for image processing

The highly-parallel integrated circuits and system (HiPIC) concept is used as the system model for real-time image processing. HiPIC is a highly-parallel system model devised for application-specific systems, to achieve both high levels of performance and flexibility. Figure 1.2 shows the basic idea of HiPIC. A HiPIC consists of a highly-parallel PE array, a reconfigurable logic element, a reduced instruction set computer (RISC) processor or a digital signal processor (DSP), and some memory. It can be implemented on a chip, as a multi-chip module, or on a board. The highly-parallel PE array (shaded block in the figure) performs SIMD processing for high-volume image data. The logic element controls the PE array and interfaces with the image data and an external processor. The processor performs serial data processing.

Various practical real-time image processing systems and their hardware algorithms, such as the facial image tracking (FIT) system [23] and the voting system [24] – [28], have already been developed based on HiPIC. Figure 1.3 shows examples of HiPIC systems. The board in the upper right of the figure is used for the real-time tracking of moving objects utilizing CAM-based local spiral labeling [23]. Labeling is one method of basic image processing, and it involves applying a unique label number to each closed region in the image. The board in the lower right of the figure is for shape and feature extraction based on the CAM-based Hough transform, and 3D projection, which are both voting methods. It can perform line and circle detection [24] – [27] and 3D feature extraction [28]. They are implemented as add-on boards for personal computers, which makes them

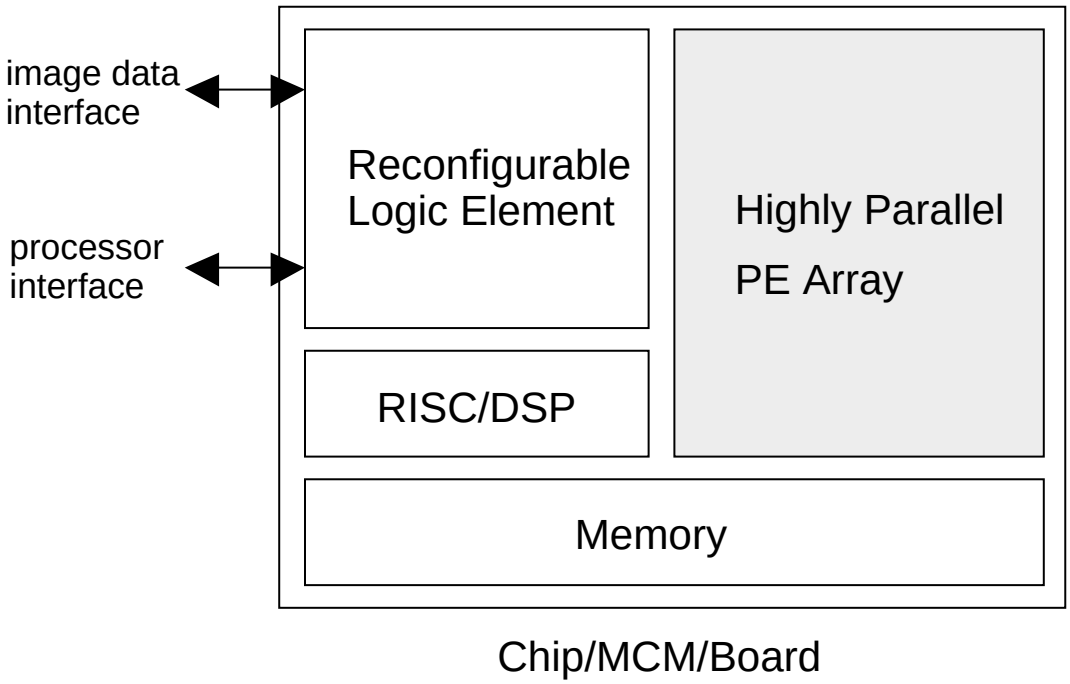


Figure 1.2: Basic idea of HiPIC.

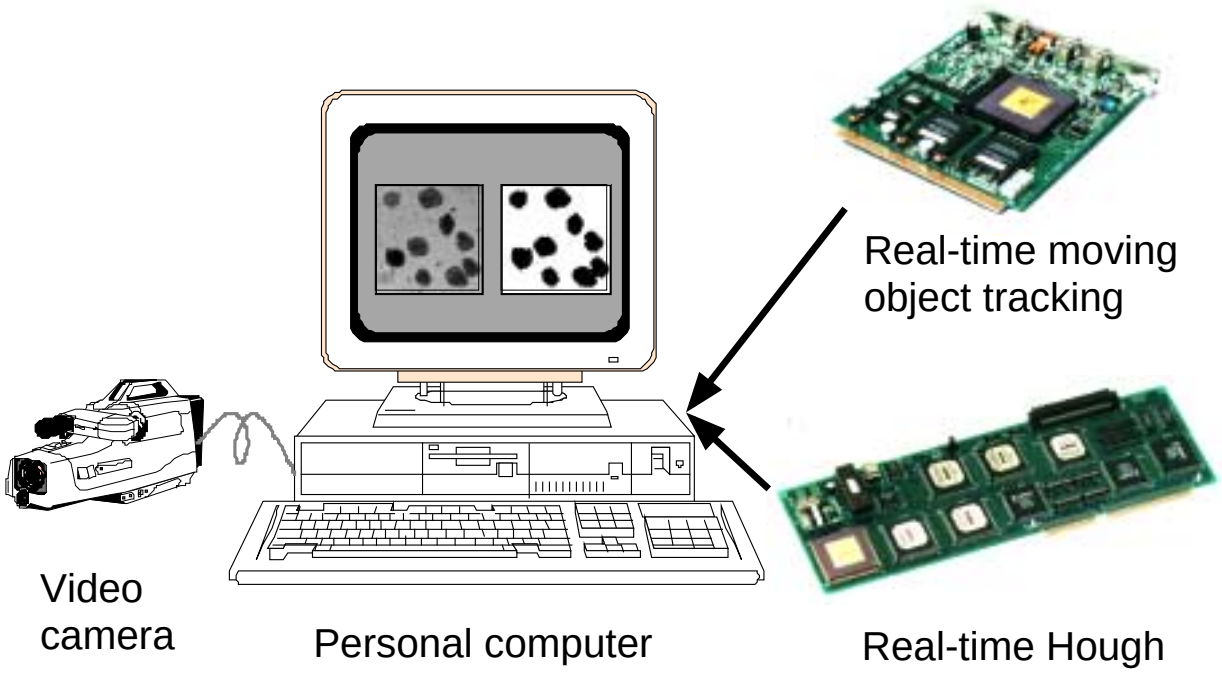


Figure 1.3: Image processing systems on HiPIC.

easy to use. These systems demonstrate that HiPIC is a very effective way to handle real-time image processing, which is why I also employed it as a system model for CAM^2.

1.2.2 LSI architecture for CA cell array

The other issue is the LSI architecture for a CA cell array. The performance of CAM^2 depends strongly on the data processing capabilities of the PE or CA cell array. The most important thing is that the array must have sufficient parallelism implemented in very little hardware. For example, several hundred thousand PEs are required for practical image processing. That means that the PE has to be simple. In addition, to avoid the von Neumann bottleneck, the most suitable PE is the logic-in-memory type. Based on these considerations, I can think of a CAM turned to CA processing as a PE array of CAM^2 on HiPIC.

Next, I survey related LSI architectures targeted for image processing. Then, I address the features and problems of conventional CAMs by comparison with these architectures.

Media processor chip

One promising LSI for creating a compact high-performance image-understanding system is a media processor [29], [30] or a microprocessor with multimedia extensions (MMX) [31], [32] which has dedicated parallel operations for image processing. This LSI is based on microprocessor technologies, and it is easy to make a high-frequency chip. However, the degree of parallelism is not very high. This prevents the full use of the abundant parallelism of pixel-parallel algorithms; and, as a result, the processing speed is insufficiently high many real-time applications.

Linear array processor chip

The degree of parallelism of a linear array processor [33], [34] is relatively high because of its simple and regular structure. However, since it can only perform one-dimensional processing (line data can be handled in parallel), iterative operations (in proportion to image size) are required to develop an entire image. By adopting a higher-performance

processing element (PE) like IMAP [33] with 8-bit-width PEs, the processing performance can be boosted. However, since the bit width of each PE is fixed, it is less flexible as regards the bit width of processed data. For example, its processing power cannot be fully utilized for a binary image.

Analog vision chip

An analog vision chip [35] – [37], which features imaging devices, such as photo detectors, and computational circuits on a single chip, is another candidate. However, it is extremely difficult to develop a large-capacity chip that can process a practical image of several hundred thousand pixels because of problems with reliability, accuracy, and production yield. Therefore, most vision processors contain very few elements, and since they do not have scalability, even if plural chips are used, they cannot handle a larger (real-world) image. Furthermore, since each element has extremely limited functions, there is insufficient flexibility for practical, or complex, image processing.

Content addressable memory (CAM)

Content addressable memory (CAM) is well known as a functional memory [38] – [41] and various types of CAM [42] – [49] have been developed since about 1980. The development trends for fully-parallel CAMs, excluding ROM and EEPROM types, are shown in Fig. 1.4.

The CAM capacity is proportional to the progress of LSI technology because CAM's memory-based structure is the most suitable for implementation with LSI technology. Indeed, as shown in Fig. 1.4, CAM capacity is increasing almost exponentially and this trend will last until the current progress on LSI process technology saturates. Moreover, its high-density structure enables us to reduce the wiring capacitance, which is the main factor behind rising power consumption. Therefore, a high capacity LSI can be implemented without causing a power consumption bottleneck.

Conventional CAMs fall into two categories:

1. CAM with only a search function, and

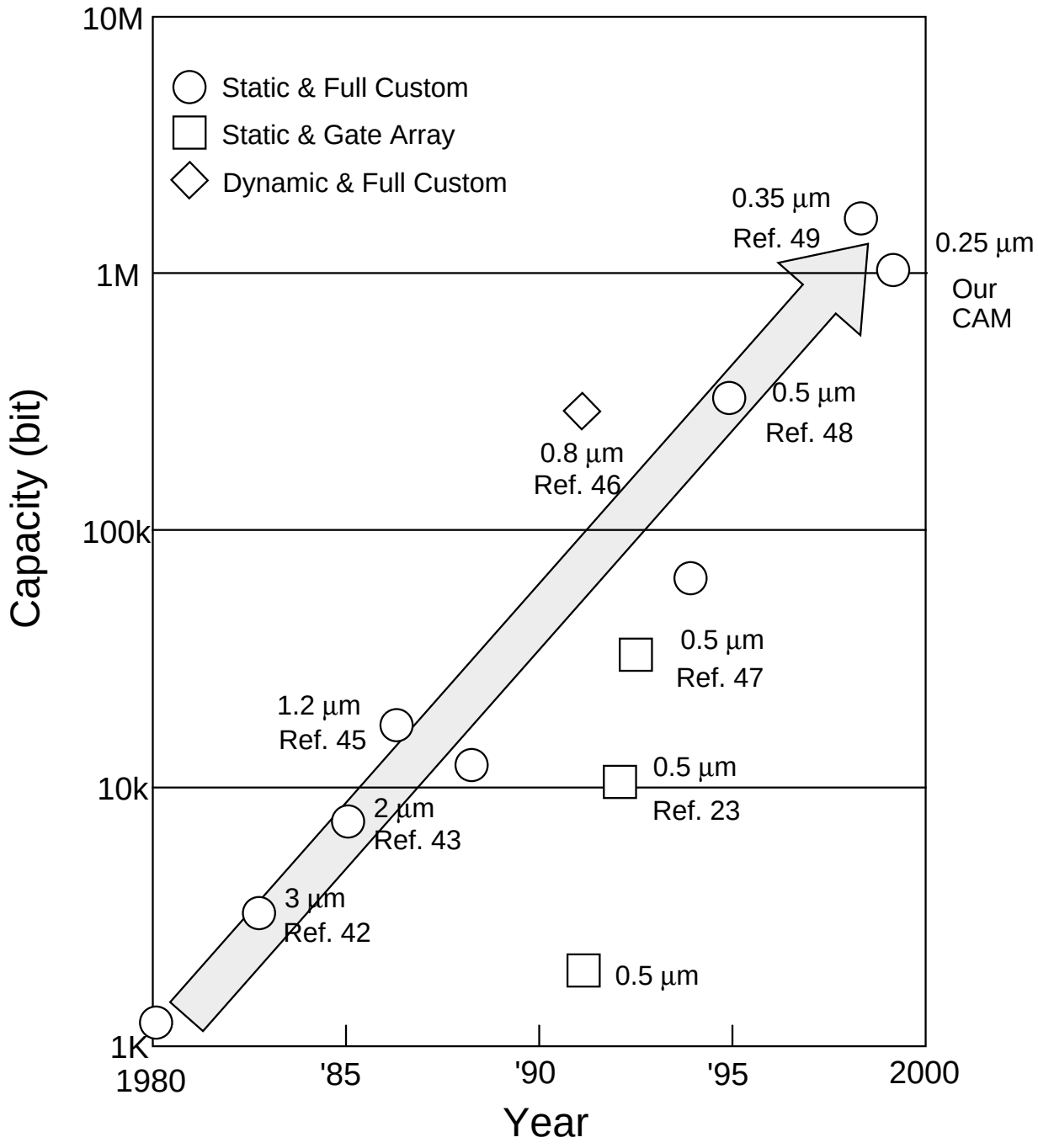


Figure 1.4: CAM development trend.

2. CAM with a parallel write function in addition to a search function.

Although the former type, which accounts for most of the conventional CAMs, has a particularly high capacity (e.g. the 2.5 Mb CAM [49]), it has only been used as a search engine in a LAN or for similar applications because of its limited functionality. In contrast, the latter type of CAM, which is also called an associative processor or a content addressable parallel processor [50], is capable of not only a parallel search but also various types of parallel data processing with words as the basic unit. This makes it a promising candidate for creating a compact, highly-parallel image processing system that requires both high levels of performance and high data throughput.

One example of an associative processor for image processing is the content addressable array parallel processor (CAAPP) [51]. It incorporates advanced communication networks, such as coterie networks, to allow it to perform a wider range of low-level computer vision applications. But only 256, or 16×16, PEs are embedded in one VLSI chip. Therefore, enormous numbers of chips are required to realize pixel-order parallelism. Moreover, since its network structure becomes extremely complicated as the array size is increased, it is difficult to increase the number of PEs even if we use state-of-the-art LSI technology. The 336 k-bit CAM [48] and the pixel-parallel image processor [52] are examples with a relatively large number of PEs (4096 PEs) reported to date. However, the 336 k-bit CAM does not have an efficient mechanism for transferring data between words, while the image processor has no functions for global data handling. This lack of flexibility severely limits their usefulness for pixel-parallel image processing applications. Moreover, they still have insufficient capacity; many chips, 16×64, are required to achieve pixel-order parallelism (several hundred thousand pixels).

1.3 Thesis scope

The objective of this dissertation is to describe the development of a compact, high-performance, flexible highly-parallel two-dimensional cellular automaton for real-time

image-understanding processing. To attain this goal, this thesis focuses on three areas of research:

1. computer architecture,
2. LSI and system design and implementation, and
3. applications.

To cover these three areas, the thesis consists of six chapters as shown in Figure 1.5.

Chapter 2 describes the basic architectural concepts and technologies of a highly-parallel two-dimensional cellular automaton architecture called CAM^2. CAM^2 is established on a system model called HiPIC and on CAM technologies. Both are vital for realizing a compact, high-performance, and flexible two-dimensional CA. The main theme of my CAM^2 study effort focused on the following three architectural considerations: CA mapping, CA processing, and data loading and retrieval processing. Multiple-zigzag mapping enables two-dimensional CA cells to be mapped into CAM words, even though physically a CAM has a one-dimensional structure. Dedicated CAM functions, such as the shift up/down mode for hit flag, enable high-performance CA processing. Furthermore, parallel loading and partial word retrieval techniques enable high throughput data transfer between CAM^2 and the outside image buffer. Chapter 2 also presents various results of a performance evaluation based on CAM^2, such as possible CAM^2 in a single PC board, and CA processing and data loading and retrieval processing performance.

Chapter 3 describes a fully-parallel 0.25 μm 1-Mb CAM LSI with dedicated functions for CA processing and a prototype CAM^2 PC board using this CAM chip. To make such a large capacity CAM that satisfies the extremely severe design constraints of state-of-the-art process technology (0.25 μm), this study involves not only VLSI circuit design, but also packaging technology, circuit board fabrication technology, power and signal distribution techniques, heat dissipation problems and design and verification strategy. Concretely, I devise a scheme that combines one-dimensional (intra-block) and two-dimensional (inter-block) physical structures and comb data and clock distribution techniques. Moreover,

I propose both manual and synthetic design strategies to speed up chip development while achieving a CAM with a high-density structure. Chapter 3 also presents various estimation results, related to such factors as operating frequency, power consumption and chip size, based on an actual fabricated chip.

Chapters 4 and 5 describe image processing applications of CAM^2 to demonstrate that CAM^2 has wide applicability to various kinds of practical real-time image processing. Many basic image processing algorithms based on CA, such as segmentation, skeletonization and cellular filtering, have already been proposed [2] and it is clear that CAM^2 can also handle them efficiently. Here, I focus on two advanced computation paradigms based on CA. These two paradigms are discrete-time cellular neural networks (DTCNN) [9], [10] and mathematical morphology [11], [12]. DTCNN is a promising computer paradigm that fuses artificial neural networks with cellular automaton (CA) concept. Mathematical morphology is an image transformation technique that locally modifies geometric features through set operations. Both are powerful tools with various applications in the field of image processing where they are becoming very commonly used [53], [54]. Here, I study a mapping and processing method designed to perform various kinds of DTCNN and morphology processing. New mapping and processing methods achieve high-throughput complex DTCNN and morphology processing. Chapters 4 and 5 also present processing time estimation results and various kinds of image processing. CAM^2 performs practical image processing, such as pattern spectrum and multiple object tracking, through a combination of DTCNN and morphology and other algorithms.

Chapter 6 summarizes the results of these research activities from the standpoint of computer architecture, LSI and system implementation and applications. Chapter 6 also indicates the future direction of research that will further improve the performance of CAM^2 and expand its use to various kinds of vision application.

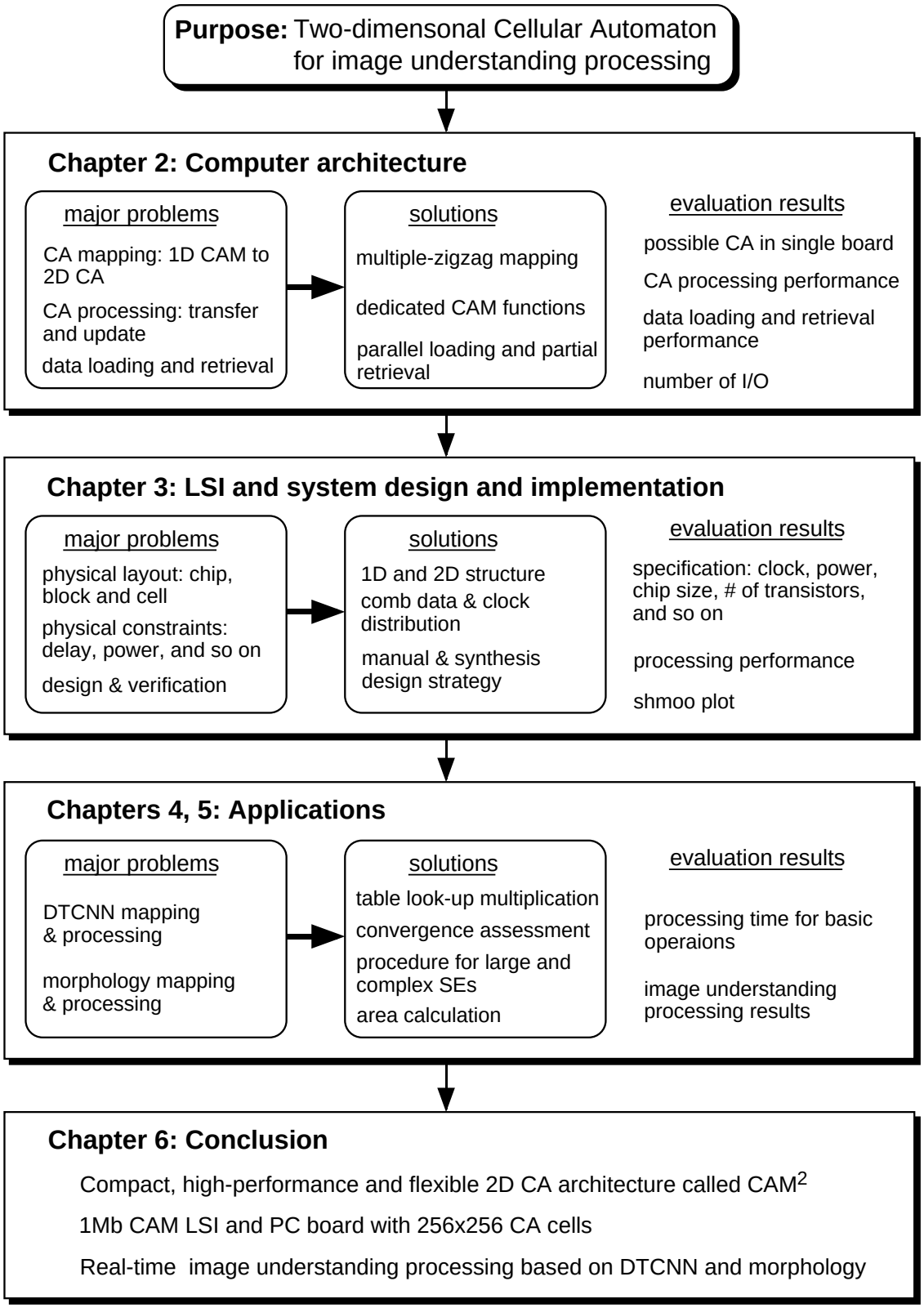


Figure 1.5: Organization of this dissertation.

Chapter 2

Cellular automaton architecture: CAM^2

2.1 Introduction

In this chapter, I explain a highly-parallel two-dimensional cellular automaton architecture called CAM^2 in detail from an architectural point of view. CAM^2 can attain pixel-order parallelism on a single PC board because it is composed of a CAM, which makes it possible to embed an enormous number PEs, corresponding to CA cells, onto one VLSI chip.

The major research subject that must be studied is mapping method of two-dimensional CA into a CAM. I devised multiple-zigzag mapping and horizontal and vertical inter-CAM connection networks for it. They enable two-dimensional CA cells to be mapped into CAM words, even though physically a CAM has a one-dimensional structure. Another important subject is how to execute CA processing on a CAM. Since conventional CAM cannot perform it, I devised dedicated CAM functions, such as the shift up/down mode for hit flag. These functions are carefully chosen not to lose CAM's simplicity, which is the most important factor to create a high-density CAM. I also propose Intra- and inter-CAM transfer procedures for CA-value transfer. These functions and procedures enable high-performance CA processing. Data loading and retrieval processing, which are rather common problems for most parallel processors, is also an important subject to study. I propose parallel loading and partial word retrieval techniques that are fully utilized CAM's feature to solve this problem. They enable high throughput data transfer

between CAM^2 and the outside image buffer.

I also show various results of a performance evaluation based on CAM^2 in this chapter. Possible CAM^2, such as CA cell size and number of I/O pins, in a single PC board is estimated based on the trend of state-of-the-art VLSI and CAM technologies. Furthermore, I present results of CA processing and data loading and retrieval processing performance. For the evaluation, the functional hardware model of CAM^2 written in Verilog-HDL [55] was developed.

This chapter is organized as follows. Section 2.2 presents the basic architecture of CAM^2. This is followed, in Section 2.3, by a description of the CA processing. After discussing the data loading and retrieval techniques in Section 2.4, the performance evaluation results are shown in Section 2.5.

2.2 Basic architecture

As described in Chapter 1, CAM^2 is established on a system model called HiPIC (highly-parallel integrated circuits and system) and on CAM technologies [22], [45]. Both are vital for realizing the compact, high-performance, and flexible two-dimensional CA. First of all, I explain how these technologies apply for creating CAM^2.

2.2.1 CAM^2 based on HiPIC

Figure 2.1 shows features of CAM^2 on HiPIC.

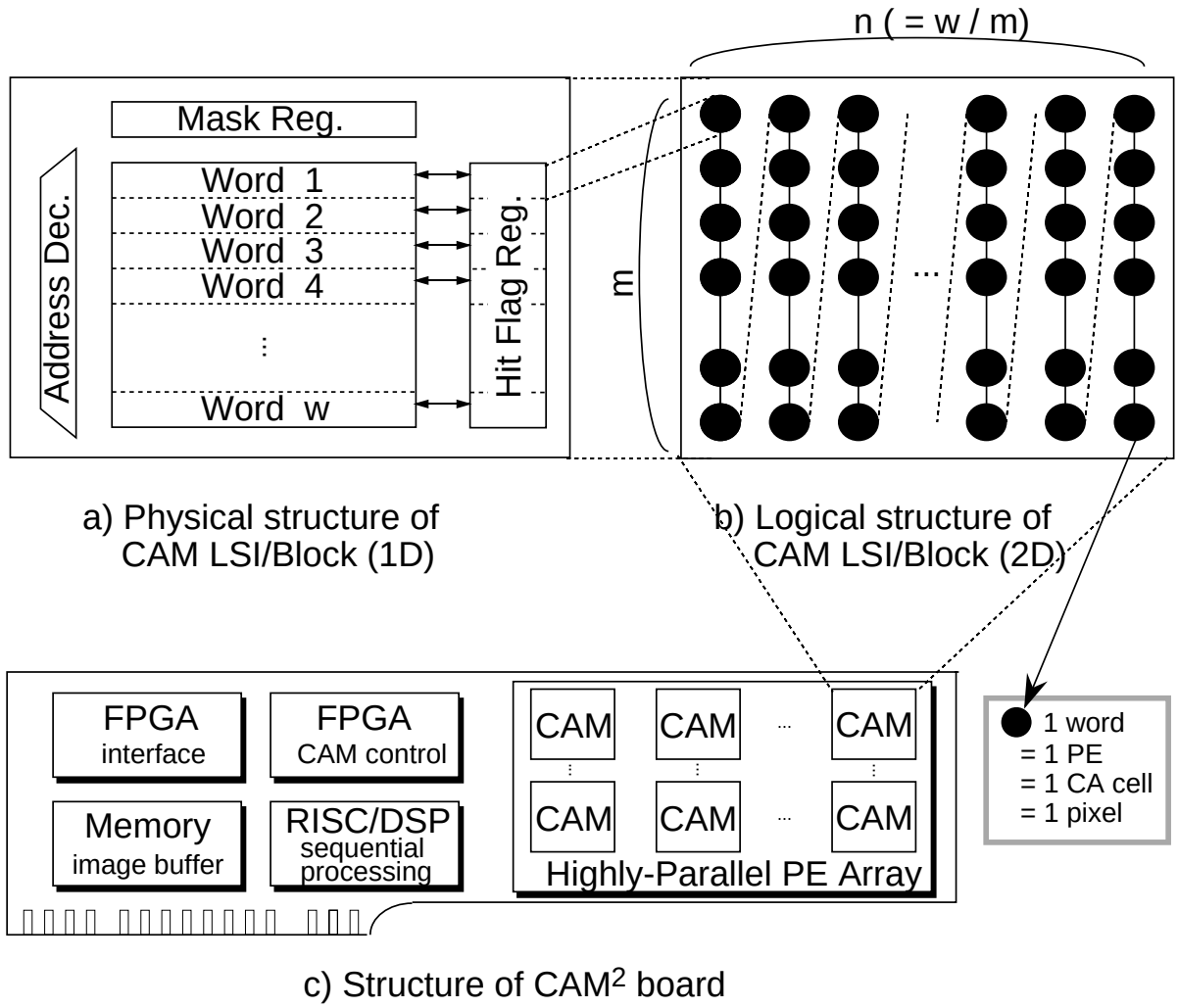


Figure 2.1: Features of CAM^2 on HiPIC.

According to the HiPIC concept, CAM^2 consists of a highly-parallel PE array, a field programmable gate array (FPGA), a RISC processor or DSP, and some memory. The PE array, which is the most important component in CAM^2, is a two-dimensional array of the proposed CAMs. Its main features are CAMs with dedicated functions for CA processing and multiple-zigzag mapping. The dedicated CAM functions enable high-performance CA processing, while the multiple-zigzag mapping enables two-dimensional CA cells to be mapped into CAM words, even though physically a CAM has a one-dimensional structure as shown in Fig. 2.1. The dedicated CAM and the PE array with multiple-zigzag mapping will be explained in detail in Sections 2.2.2 and 2.2.3, respectively.

The FPGA controls the highly-parallel PE array. The control logic, which generates command sequences for CA processing, is mapped into the FPGA. Since an FPGA can easily rewrite the logic, CAM^2 performs various types of CA-based image processing, such as discrete-time cellular neural networks [9], [10], mathematical morphology [11], [12], linear filtering [7], either alone or in combination. (The processing methods of DTCNN and morphology will be described in detain in Chapters 4 and 5, respectively.) Furthermore, the interface logic is also mapped into the FPGA. By changing the logic, CAM^2 can be embedded into various computer platforms.

The processor performs serial data processing. Since it can be used for various pre- and post-CA processing, CAM^2 can handle total image processing, including higher-level processing.

The memory is used as a buffer. It stores input images, temporary data, and processed data. It is also used for storing some programs for the FPGA.

2.2.2 Dedicated CAM features for CAM^2

CA processing is carried out by iterative operations of CA-value transfer and update. To perform both operations effectively, the dedicated CAM for CAM^2 has the following functions.

- Word read/write using addresses (2.a)
- Maskable search (2.b)
- Parallel/partial write (2.c)
- Shift up/down mode for hit flag (2.d)
- Parallel execution of hit flag shift (2.d) and word read/write (2.c) (2.e)

The dedicated CAM can perform normal RAM operations, such as word reads and writes using addresses (2.a). Using (2.a), every word can be directly accessed from/to Data I/O. Furthermore, it can also operate as a SIMD PE array and do such things as a

maskable search (2.b), and partial and parallel writes (2.c). For the search, the results are accumulated in hit-flag registers by means of OR logic. For the writes, the data are written into specific bit positions of multiple words for which the value of the hit-flag register is 1. Through the iteration of these two operations, the SIMD-type PE operations, such as general associative, logical and arithmetic operations, can be carried out in a bit-serial, word-parallel manner.

Our previously developed CAMs [45], [22] also have functions (2.a)-(2.c). However, the data transfer between neighboring CA cells, which is indispensable for CA processing, cannot be performed using only these functions. Therefore, two dedicated operations are added: upward and downward hit-flag shift (2.d), and the parallel execution of a hit-flag shift and a read or write using an address (2.e). These functions can be implemented just by changing the peripheral circuit of CAM, i. e., changing the memory cell part is unnecessary. So, the dedicated CAM LSI for CAM^2 can be easily built by making minimum refinements to our previously developed CAMs, such as the 0.5-μm 336 k-bit CAM [48].

Figure 2.2 shows a block diagram of the dedicated CAM for CAM^2. It consists of five function blocks: a cell array block, a bit operation block, a word operation block, an interface block and a control block.

The cell array block is composed of an address decoder and words whose number is w. The word is divided into writable bits and read-only bits. The writable bits can perform both the maskable search (2.b) and the parallel write (2.c). So, SIMD-type PE operations can be carried out to data on the bits. On the other hand, the read-only bits store fixed address data and can only perform the maskable search (2.b) for the data. By masking the read-only bits, plural words can be addressed.

The bit operation block controls the bit position for the maskable search (2.b) and the parallel write (2.c). It is composed of a search mask register (SMR) and a write mask register (WMR). SMR stores search mask data for both the writable and read-only bits. WMR stores write mask data for only the writable bits.

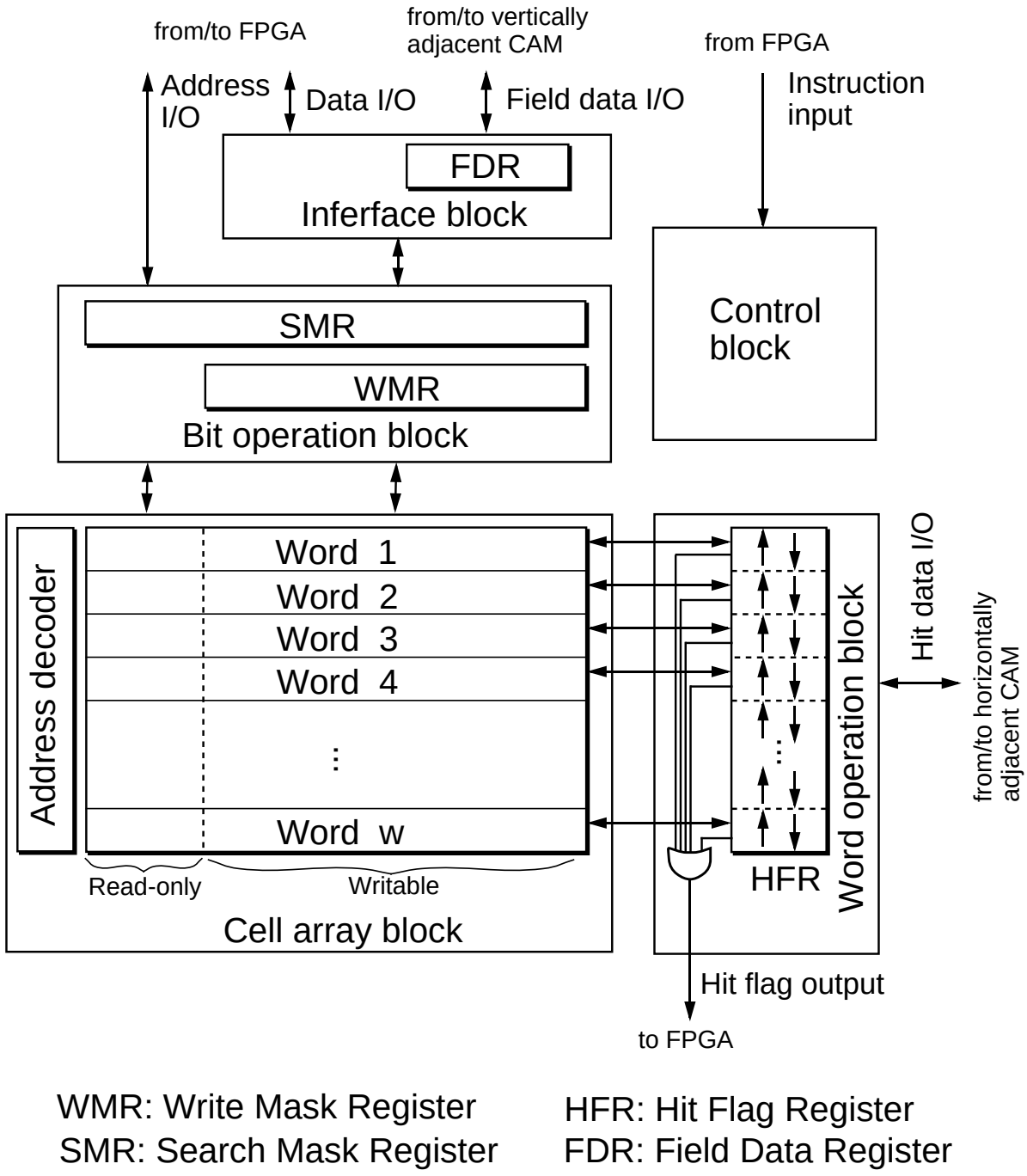


Figure 2.2: Block diagram of the dedicated CAM.

The word operation block is composed of hit-flag registers (HFR) which store the hit flags of all words. Using (2.d), the hit data are shifted to upper or lower words. So, the hit-flag register can be used as a 1 bit one-dimensional data transfer path between words. These data can be shifted in/out through the Hit Data I/O. The Hit Data I/O is connected to horizontally adjacent CAMs.

The interface block switches the path from/to the CAM controller (FPGA) and the path from/to vertically adjacent CAMs. It has a field data register (FDR) which stores the field data of one word. FDR is used in the CA-value transfer operation.

Since all of the CAM^2 functions (2.a)-(2.e) are very simple, it is easy to implement. Moreover the physical structure is very similar to that of a random access memory, as shown in Fig. 2.2, and this type of memory-based structure is very suitable for implementation with LSI technology. Therefore, I can make an enormous number of PEs with state-of-the-art process technology. The drawback of this scheme is that complex operations, such as the multiplication of many bits, take a long time. Moreover, CAM^2 cannot perform floating-point operations. However, since most CA-based algorithms do not need such complex operations, I think the above functions will be useful for a variety of CA operations.

2.2.3 Highly-parallel PE array with multiple-zigzag mapping

Figure 2.3 shows the basic structure of the highly-parallel PE array.

The PE array consists of a two-dimensional array ($q \times r$) of the proposed CAMs. Prominent features of the configuration are as follows:

- Multiple-zigzag mapping (2.f)
- Horizontal inter-CAM connection networks (2.g)
- Vertical inter-CAM connection networks (2.h)

As mentioned before, CAM has only a one-dimensional data transfer path. Hence, zigzag mapping is necessary in order to assign a two-dimensional CA ($X \times Y$) to CAM

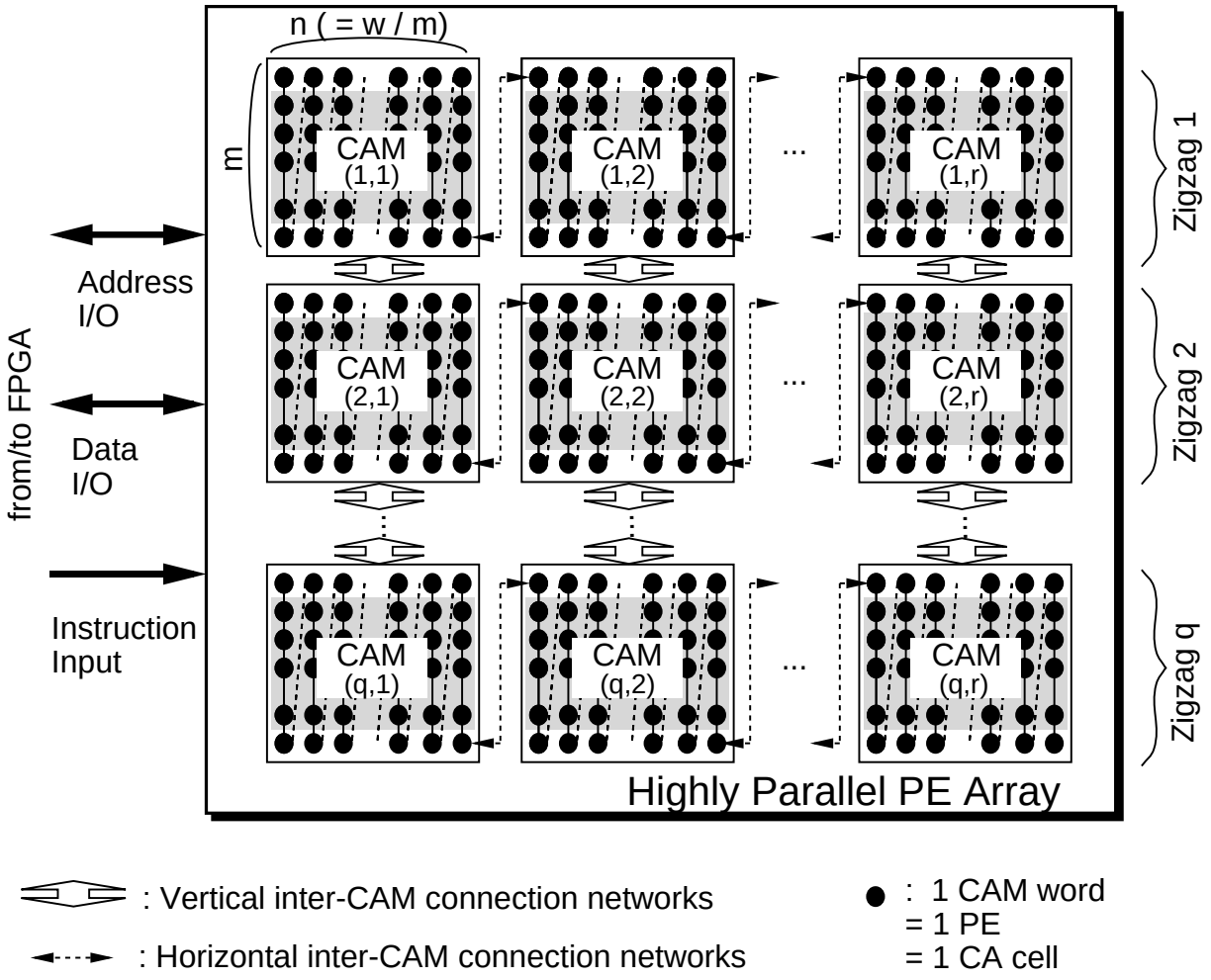


Figure 2.3: Configuration of the highly-parallel PE array.

word. If single-zigzag mapping is used for this purpose, transferring the value from the original cell to the horizontally adjacent cell requires many shift cycles, as shown in Fig. 2.4(a). To address this problem, multiple-zigzag mapping (2.f) is devised. Since multiple-zigzag mapping enables us to reduce the distance between horizontally adjacent cells ($\frac{1}{q}$ that of single-zigzag mapping), as shown in Fig. 2.4(b), it reduces transfer cycles significantly.

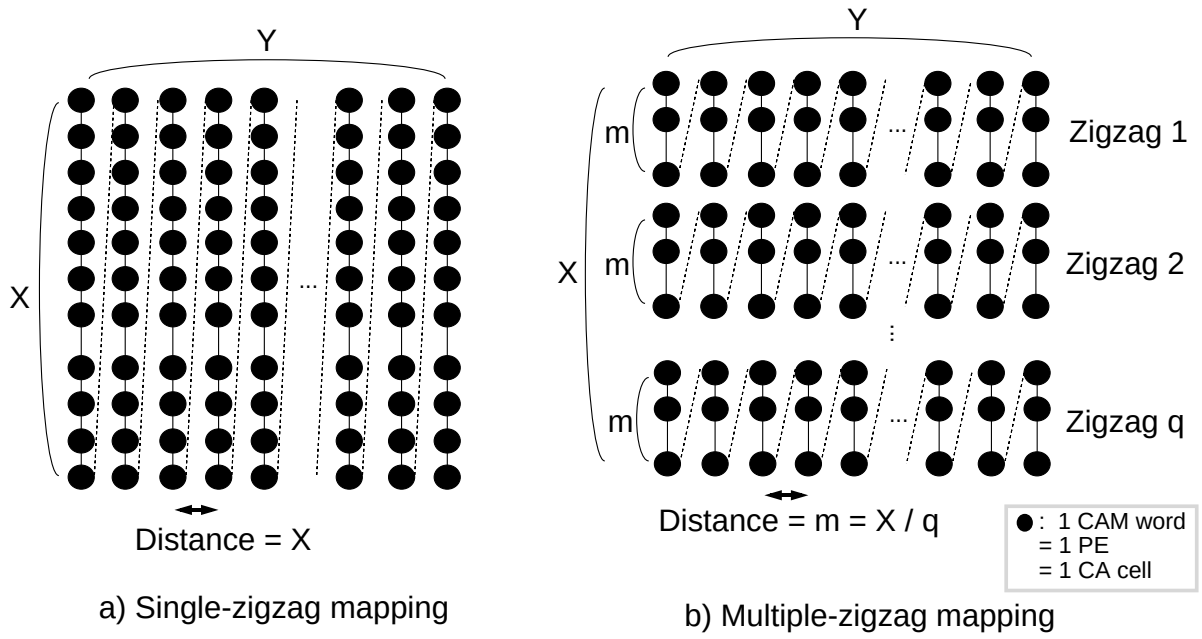


Figure 2.4: Comparison of single and multiple zigzag mapping.

According to the strategy, the w words of each CAM are segmented every m words and mapped to $m \times n$ CA cells. Totally, a two-dimensional CA having $(X = m \times q) \times (Y = n \times r)$ CA cells is realized, as shown in Fig. 2.3.

The horizontal inter-CAM connection networks (2.g) connect the hit-flag registers of horizontally adjacent CAMs through the Hit Data I/O. This enables the horizontally adjacent CAMs to be considered one CAM in the hit-flag register shift mode (2.d). Therefore, data transfer between horizontally adjacent CAMs can be completely performed using (2.f) and (2.g).

On the other hand, vertical data transfer for boundary words, the unshaded parts of the

CAM in Fig. 2.3, can not be performed in above processing. So, multiple-zigzag mapping (2.f) is not applicable to a conventional one-dimensional PE array architecture. But a CAM can directly access each word through normal RAM operations (2.a). Therefore, using this function and the vertical inter-CAM connection networks (2.h), which connect the vertically adjacent CAMs through the Field Data I/O, the vertical data transfer of the boundary words can be performed, too.

Furthermore, the connection networks scheme combining the one-dimensional intra-CAM and the two-dimensional inter-CAM (2.g) (2.h) holds down the increase rate of I/O pins. Therefore, the high level of performance and compact PE array can be realized without I/O bottlenecks. The evaluation result for I/O pins is discussed in Section 2.5.

2.3 CA processing

In this section, CA processing using CAM² is explained. As an example of the processing, a four-neighbor (left, right, up, and down) CA model is used.

Figure 2.5 shows the CAM word configuration for the CA model.

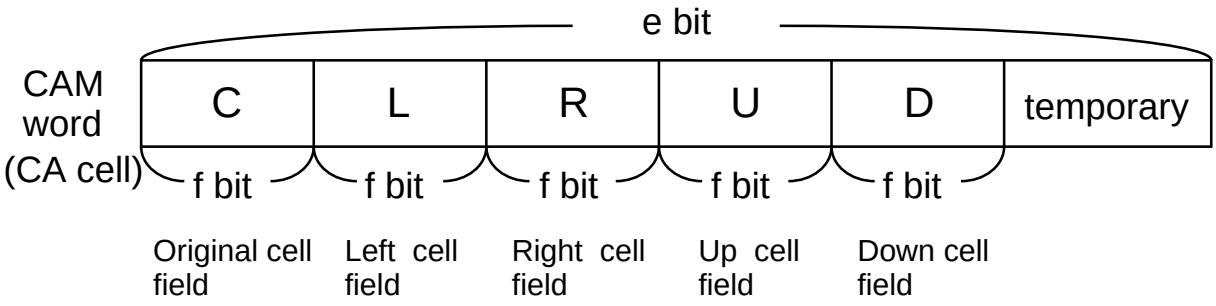


Figure 2.5: CAM word configuration for a four-neighbor CA model.

Each CAM word, whose bit width is e, consists of an original cell value field, neighbors' cell value fields and a temporary field. The original cell field (C) stores the value of the original cell. The neighbors' cell fields store values in the left (L), right (R), up (U), and down (D) cells. The bit width of these fields is f, where f is determined by the dynamic range of the CA value. The temporary field is used for storing carry, flag, and so on. All

fields are assigned to the writable bits.

CA processing is carried out by the iterative operations of CA-value transfer and update. A transfer operation involves transferring the (C) values of neighboring cells to the corresponding (L), (R), (U), and (D) fields. An update involves calculating the next value for the original cell by applying a particular transition rule to the (C), (L), (R), (U), and (D) values. These operations are completely carried out in parallel.

2.3.1 CA-value transfer

In the CA processing, the CA-value transfer process dominates the performance of CA processing in CAM^2. To improve the performance, I devised two types of CAM transfer: intra-CAM and inter-CAM transfer. In the intra-CAM and inter-CAM transfer processes, inner words and boundary words are transferred, respectively.

Intra-CAM transfer

The intra-CAM transfer is carried out in the following sequence:

1. Set search mask.
2. Set write mask.
3. Maskable search to a certain bit of (C).
4. Upward/downward shift of hit flag (1 bit).
5. Parallel writing to the correspondent bit position of neighbors' cell field (U/D).
6. Upward/downward shift of hit flag ($m-1$ bits).
7. Parallel writing to the correspondent bit position of neighbors' cell field (R/L).
8. Repeat 1-6 until all bits (f bits) of (C) are transferred.

As this shows, the processing is carried out in a bit serial manner. But all of the processing, including the hit-flag shift between horizontally adjacent CAMs, are carried

out in parallel as explained in Section 2.2.3. Therefore, the entire intra-CAM transfer can be done very quickly.

Figure 2.6 shows the intra-CAM transfer, in which a certain bit of original cell (C) is transferred to its right and down CA cells. In the transfer, m cycles are needed for the hit-flag shift and α cycles for the rest of the operations, where α is less than 10. So, the number of cycles needed to transfer all bits to the right and down CA cells $T_{intra(DR)}$ is

$$T_{intra(DR)} = (m + \alpha) \times f. \tag{2.1}$$

The intra-CAM transfer to the left and up CA cells is done in the same way using the hit-flag shift-up function. Therefore, the total number of cycles for the inter-CAM transfer T_{intra} is

$$T_{intra} = (m + \alpha) \times f \times 2. \tag{2.2}$$

In the intra-CAM transfer, the hit-flag shift is the most time-consuming step. However, the multiple-zigzag mapping (2.f) reduces the number of cycles needed. For example, if the number of zigzags is q, the transfer cycle is about $\frac{1}{q}$ compared with that of single-zigzag mapping.

Inter-CAM transfer

The inter-CAM transfer is carried out in the following sequence:

1. Transfer all bits (f bits) of a certain (C) in lower boundary words to FDR.
2. Transfer the data in FDR to the correspondent (U) in upper boundary words.
3. Repeat 1-2 until all lower boundary words are transferred.
4. Transfer all bits (f bits) of a certain (C) in upper boundary words to FDR.
5. Transfer the data in FDR to the correspondent (D) in lower boundary words.
6. Repeat 4-5 until all upper boundary words are transferred.

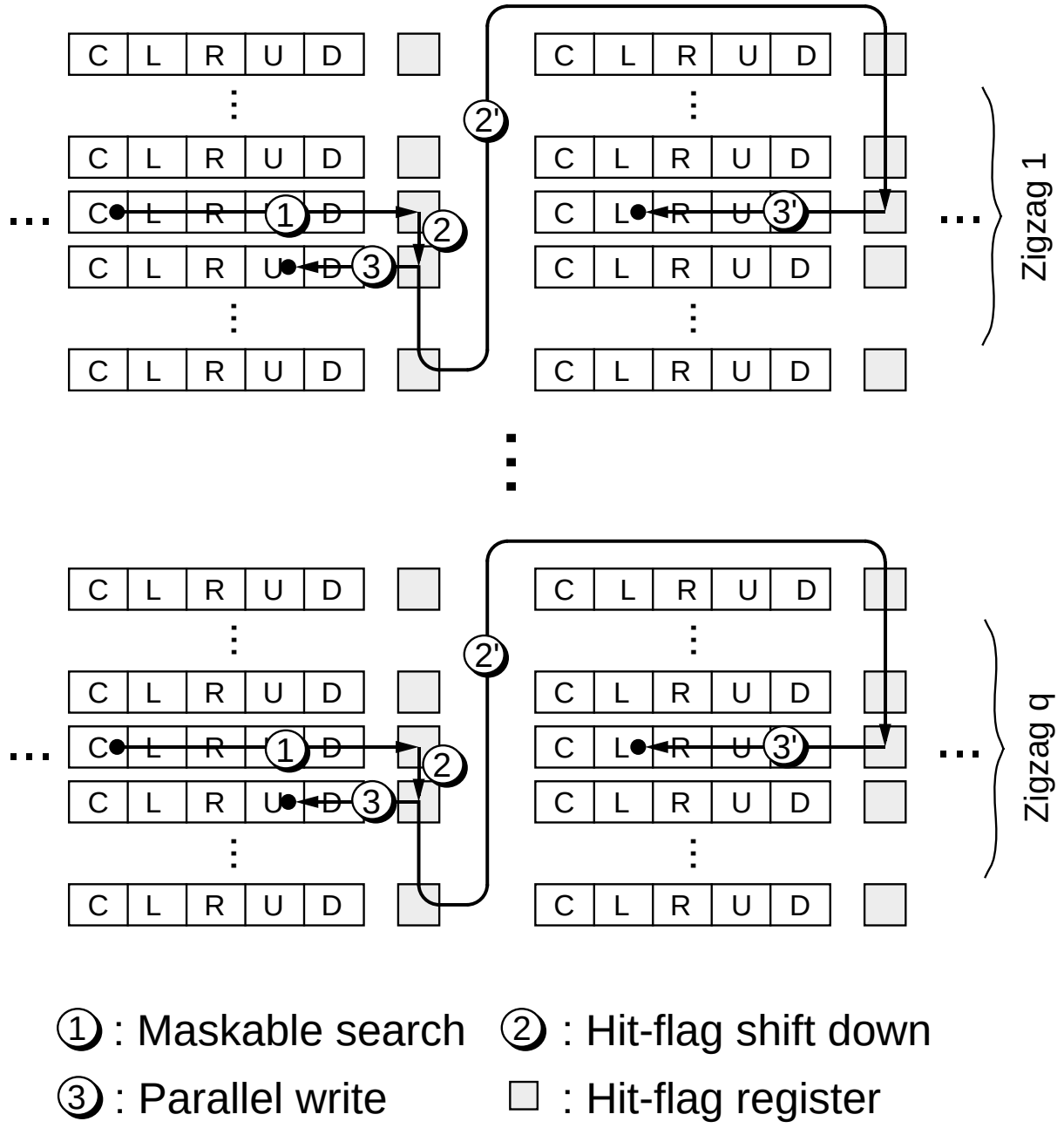


Figure 2.6: Example of intra-CAM transfer (1 bit, to the right and down CA cells).

All of the steps are carried out by the word read/write (2.a). Using the vertical inter-CAM connection networks, all CAMs can perform the word read/write (2.a) in parallel. Therefore, the inter-CAM transfer is also effectively performed.

Since the numbers of upper and lower boundary words in one CAM are both n, the total number of cycles for the inter-CAM transfer T_{inter} is

$$T_{inter} = n \times 4. \tag{2.3}$$

An example of the inter-CAM transfer from lower boundary words to upper boundary words is shown in Fig. 2.7.

Function (2.e) enables us to reduce the CA-value transfer cycle still more. Using this function, the inter-CAM transfer can be carried out simultaneously during the hit-flag shift (2.d) in the intra-CAM transfer. By choosing proper m and n, the CA-value transfer cycle can be dramatically reduced.

From the discussion above, it is clear that the dedicated CAM functions and mapping method (2.a)-(2.h) make a significant contribution to the reduction of the CA-value transfer cycles.

2.3.2 CA-value update

After the CA-value transfer process finishes, the CA-value update process is invoked. In the operation, according to the transition rule, various logical and arithmetic calculations using the original and neighbors' cell field values are carried out, and the result is stored to the original cell field (C). The maskable search (2.b) and the partial and parallel writes (2.c) are used for the CA-value update processing. Through the iteration of these two operations, various operations can be carried out.

For example, the f-bit GREATER-THAN operation $\{greater\_than(X, Y) \Rightarrow X\}$ that is required for processing gray-scale morphology (which will be explained in detail in Chapter 5) is executed in the following steps:

1. Set search mask.

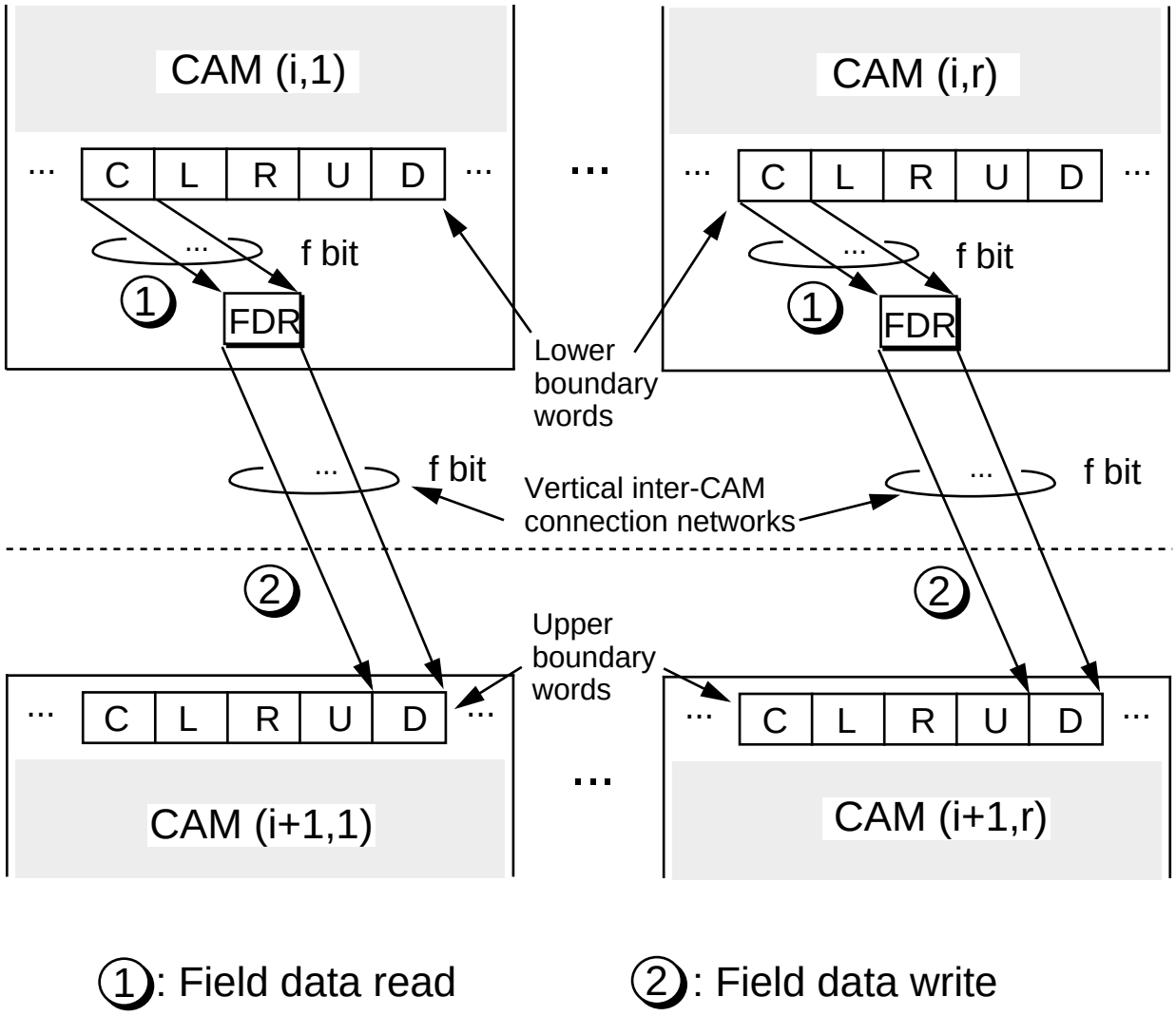


Figure 2.7: Example of inter-CAM transfer (f bit, from lower boundary words to upper boundary words.

2. Set write mask.

3. Maskable search for words whose Xi, Yi, F1, F2 is 0, 1, 1, 1.

4. Maskable OR search for words whose Xi, Yi, F1, F2 is 0, 1, 0, 0.

5. Parallel writing of 1, 0, 0 to Xi, F1, F2 of the hit words.

6. Maskable search for words whose Xi, Yi, F1, F2 is 1, 0, 1, 1.

7. Parallel writing of 1, 0, 1 to Xi, F1, F2 of the hit words.

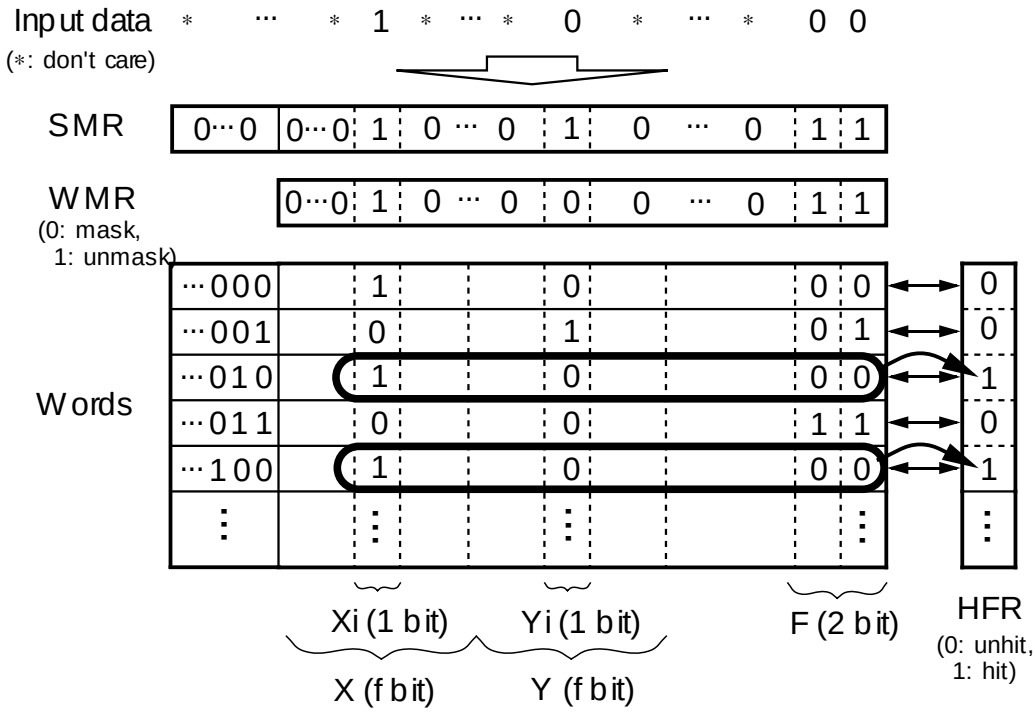
8. Maskable search for words whose Xi, Yi, F1, F2 is 1, 0, 0, 0.

9. Parallel writing of 0, 0, 0 to Xi, F1, F2 of the hit words.

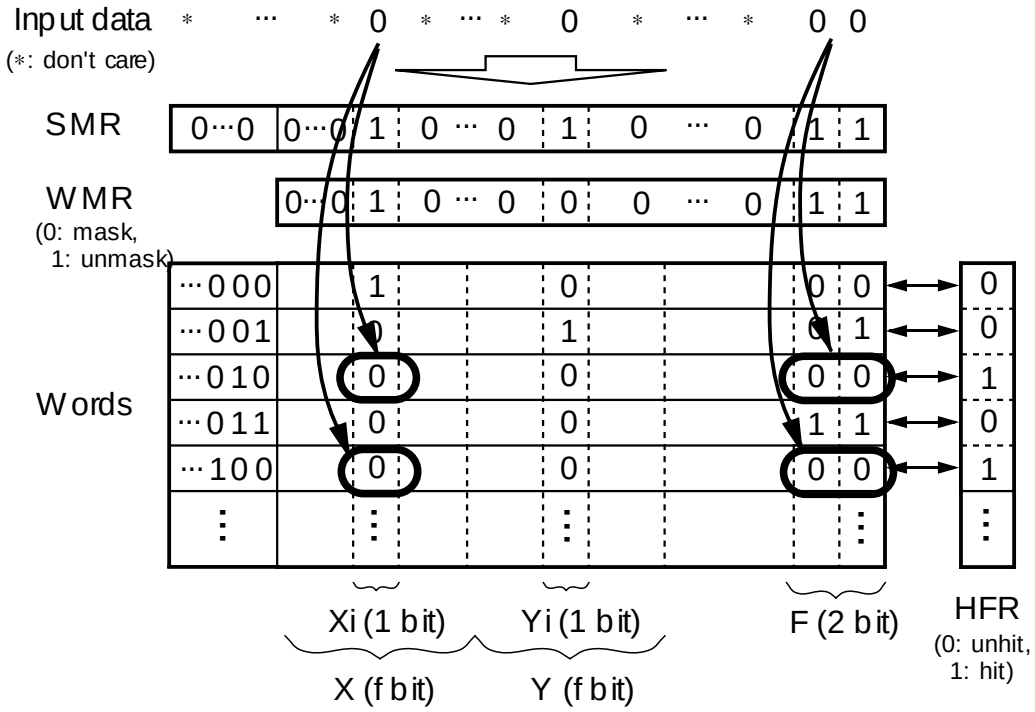
In this sequence, F1 and F2 are used as flags. (The initial values of F1 and F2 are both 1. The condition of F1, F2 = 0, 0 indicates that "Y > X" is determined and the condition of F1, F2 = 0, 1 indicates the opposite.) Figure 2.8 shows examples of the maskable search in step 8 and the parallel write in step 9.

Since the processing must be repeated from MSB (I = f) to LSB (I = 1), the processing time increases with their bit length. However, every word can perform the processing in parallel. Therefore, the update operations for all PEs are completed in an extremely short time. The number of cycles for the f-bit GREATER-THAN operation is

$$C_{greater-than} = 9 \times f. \tag{2.4}$$



a) Maskable search



b) Partial & parallel write

Figure 2.8: Example of greater than operation $\{greater\_than(X,Y) \Rightarrow X\}$.

2.4 Data loading and retrieval processing

To complete the image processing, not only the CA processing described in Section 2.3 but also data loading and retrieval processing are required. For the loading, all pixel data of input image are loaded into destination fields of the correspondent CA cells of CAM². For the retrieval, the processed data are retrieved from result fields of all CA cells.

If these types of processing are carried out one by one using word read/write (2.a), the processing time becomes extremely long. For example, more than 5 hundred thousands cycles are needed for CAM² with 512 × 512 CA cells. Therefore they dominate the performance of CAM², especially for real-time applications, in which data loading and retrieval must be performed every frame. To shorten the time, parallel loading and partial retrieval methods are devised.

2.4.1 Parallel loading

The parallel loading is carried out in two phases: block data parallel write and inter-field transfer. The parallel write is carried out in the following sequence:

1. Set search mask.
2. Set write mask.
3. Select b pixels in input image as a block data.
4. Write the block data into b words in parallel.
5. Repeat 3-4 until all pixels are selected.

In the sequence, each block data has $f \times b$ bits, where f is the bit width of pixel data. Since the block data must be written into one word, b is determined according to $b \leq e/f$, where e is the bit width of one word. The parallel write is finished in $\frac{Wall}{b}$ cycles, where $Wall$ is the number of total words.

Figure 2.9 shows an example of the parallel write. In the example, b was assumed to be 8. Since 8 words are addressed simultaneously by masking the lower read-only bits (3 bits), the parallel write in step 4 can be performed.

After the parallel write process is over, all pixel data are loaded. But most of the data exist in a temporary field that is not the destination field. Therefore the data must be transferred to the destination field. For this processing, the inter-field transfer process is invoked. The inter-field transfer is carried out in the following sequence:

1. Set search mask.
2. Set write mask.
3. Maskable search to a certain bit of a certain pixel data in the temporary field.
4. Parallel writing to the correspondent bit position of the destination field.
5. Repeat 1-4 f times.
6. Repeat 1-5 until all pixel data in the temporary field are transferred to the destination field.

After the inter-field transfer process is over, all pixel data are loaded into the destination field. Figure 2.10 shows an example of the inter-field transfer. By masking the higher read-only bits (except lower 3 bits), steps 1-4 can be performed every 8 words in parallel.

The number of cycles needed for conventional method adopting sequential writing $T_{load-conventional}$ is

$$T_{load-conventional} = Wall. \tag{2.5}$$

On the other hand, using the proposed parallel loading method, the number of cycles T_{load} is

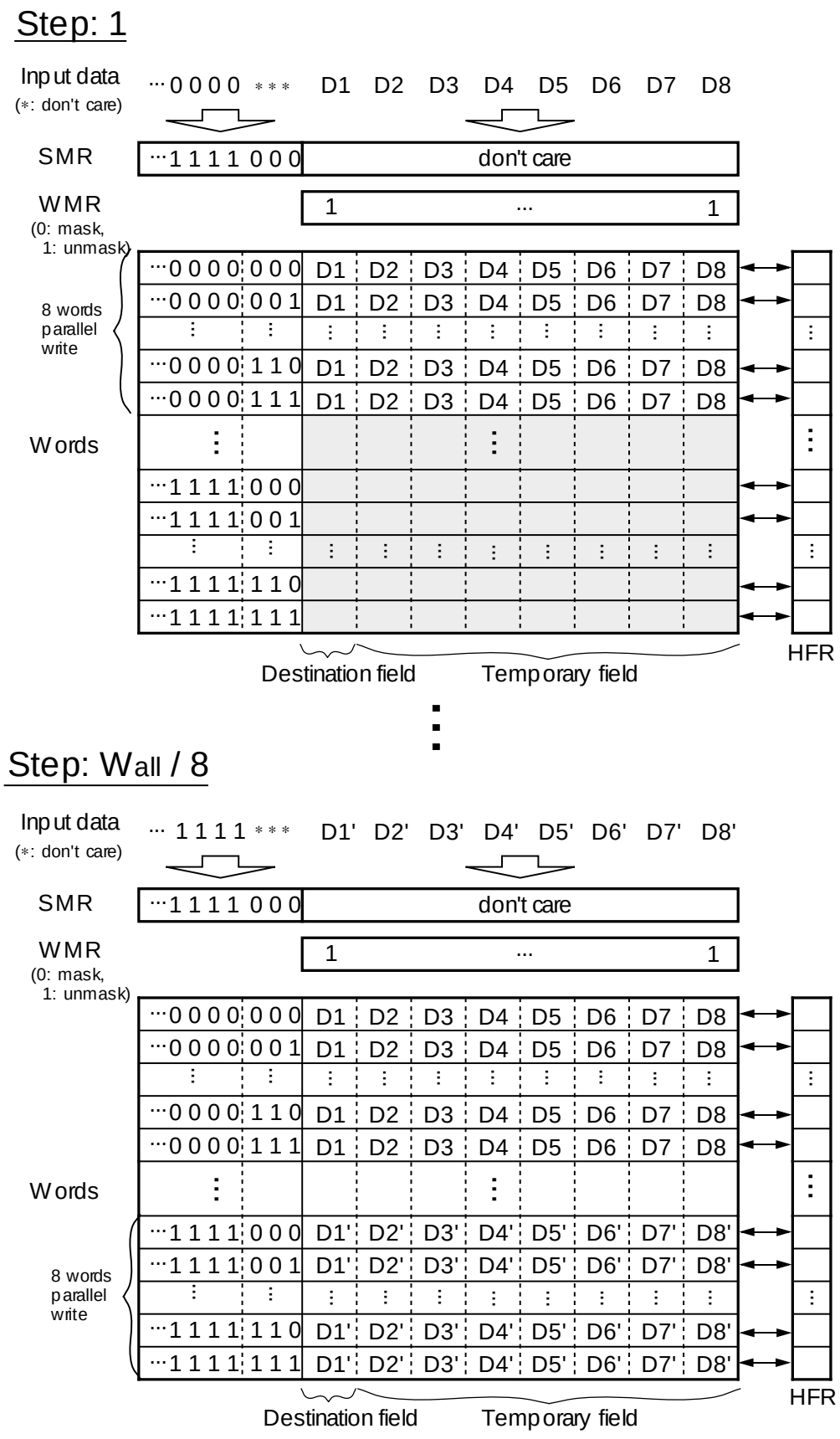


Figure 2.9: Example of block data parallel write.

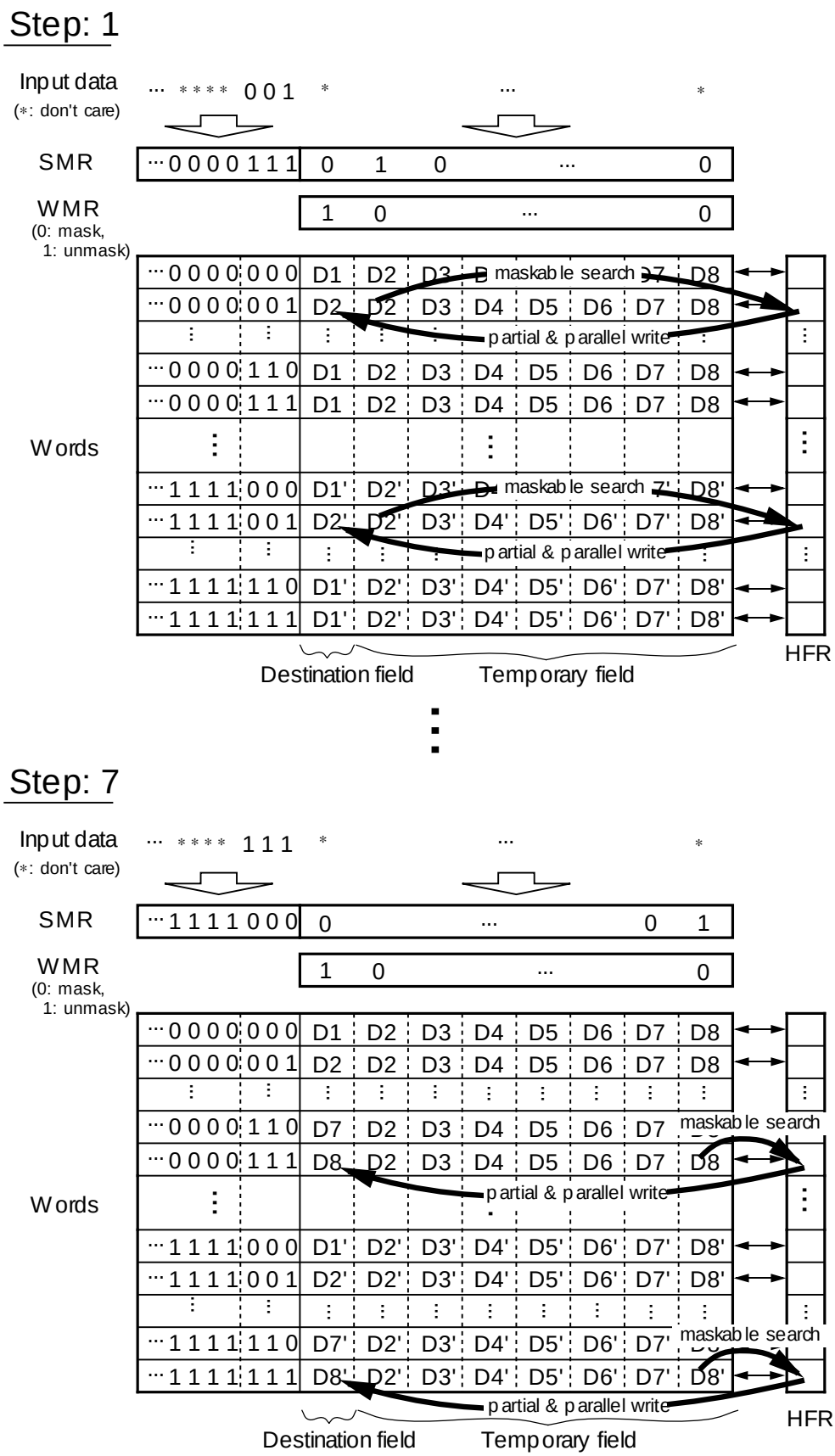


Figure 2.10: Example of inter-field transfer.

$$T_{load} = \frac{Wall}{b} + f \times \beta \times (b-1) \tag{2.6}$$

where β denotes the cycles for the maskable search and the parallel write and its value is less than 5. When $Wall$ is pixel-order (several hundred thousand), the value of the second term is very low compared with that of the first term. So, our method is about b times faster than the conventional method.

2.4.2 Partial retrieval

The partial retrieval is carried out in two phases: inter-word transfer and partial word read. The inter-word transfer is carried out in the following sequence:

1. Select partial words (every b words).
2. Set search mask.
3. Set write mask.
4. Maskable search to a certain bit of a word except the partial words.
5. Upward/downward shift of hit flag to the position of the partial word.
6. Parallel writing to the correspondent bit position of the partial word.
7. Repeat 2-5 f times.
8. Repeat 2-6 until all result data are transferred to the partial words.

Through the inter-word transfer, all data in the result field are gathered into the partial words. Figure 2.11 shows an example of the inter-field transfer. In the example, b was assumed to be 8.

After the inter-word transfer process is over, the partial read process is invoked to retrieve the result from CA cells. By reading only the partial words whose number is $\frac{1}{b}$ of all words, all result data can be retrieved.

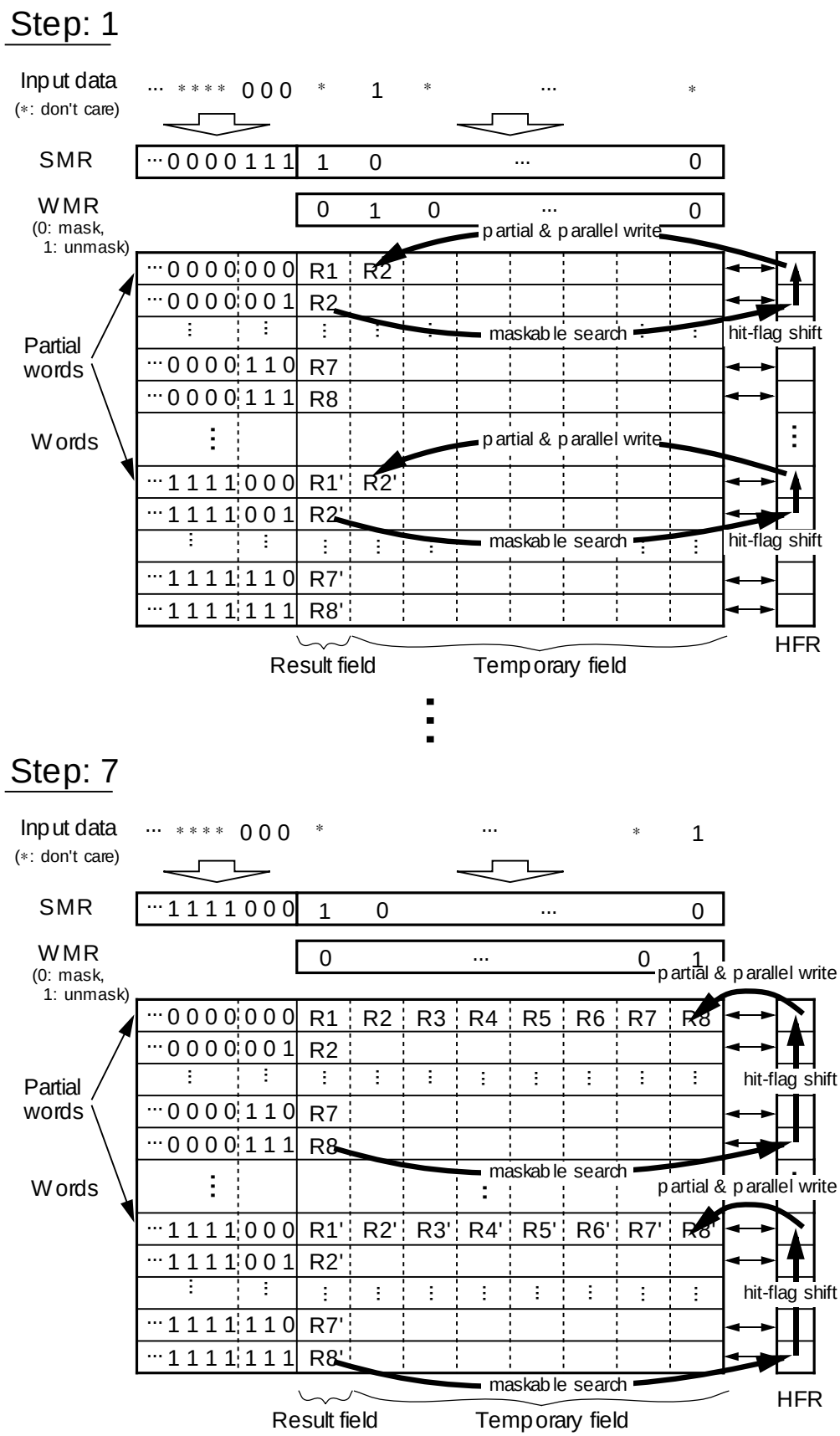


Figure 2.11: Example of inter-word transfer.

The number of cycles needed for the conventional method adopting every word reading $T_{retrieval-conventional}$ is

$$T_{retrieval-conventional} = Wall. \tag{2.7}$$

On the other hand, using the proposed partial retrieval method, the number of cycles $T_{retrieval}$ is

$$T_{retrieval} = \frac{Wall}{b} + f \times (\beta + \frac{b(b-1)}{2}) \times (b-1). \tag{2.8}$$

Since the values of the second term can also be ignored when $Wall$ is pixel-order, our method is about b times faster than the conventional method.

2.5 Evaluation

2.5.1 Possible CAM² in a single board

For the purpose of building a compact and practical CA, CAM² should be developed in a single personal computer (PC) board. On a single PC board, up to sixteen CAM LSIs can be embedded. Table 2.1 shows the estimation of possible CAM² using 16 CAM LSIs.

Table 2.1: Possible CAM² in a single PC board.

CMOS tech.	Total CAM bit (1 CAM × chip #)	CA cell size (= pixel size)	
		for gray scale image	for binary image
0.5 μm	256 k × 16	256 × 256 (1 word = 64 bits)	512 × 512 (1 word = 16 bits)
deep sub-micron	1 M × 16	512 × 512 (1 word = 64 bits)	1024 × 1024 (1 word = 16 bits)

A 336 k-bit CAM LSI has already been developed using 0.5-μm CMOS technology [22]. According to process trends, a 256-kbit and a 1-Mbit dedicated CAM LSI for CAM² can also be developed using 0.5-μm and deep sub-micron CMOS technology, respectively. Using 0.5-μm CMOS technology, a 256 × 256 CAM² can be realized by assigning 64 bits

to one word. This bit length is sufficient for gray-scale image processing. If 16 bits, which are sufficient for binary image processing, are assigned to one word, a 512 × 512 CAM^2 can be achieved. Moreover, using deep sub-micron CMOS technology, the size of the cell embedded in a single PC board is increased by a factor of four. Hence, using state-of-the-art CMOS technology, it is feasible that a compact (a single PC board) and practical (256 × 256 – 1024 × 1024 pixel-level image processing) CAM^2 can be made.

Figure 2.12 shows the estimation of the number of signal I/O pins of the dedicated CAM LSIs with various numbers of words (1 k - 256 k).

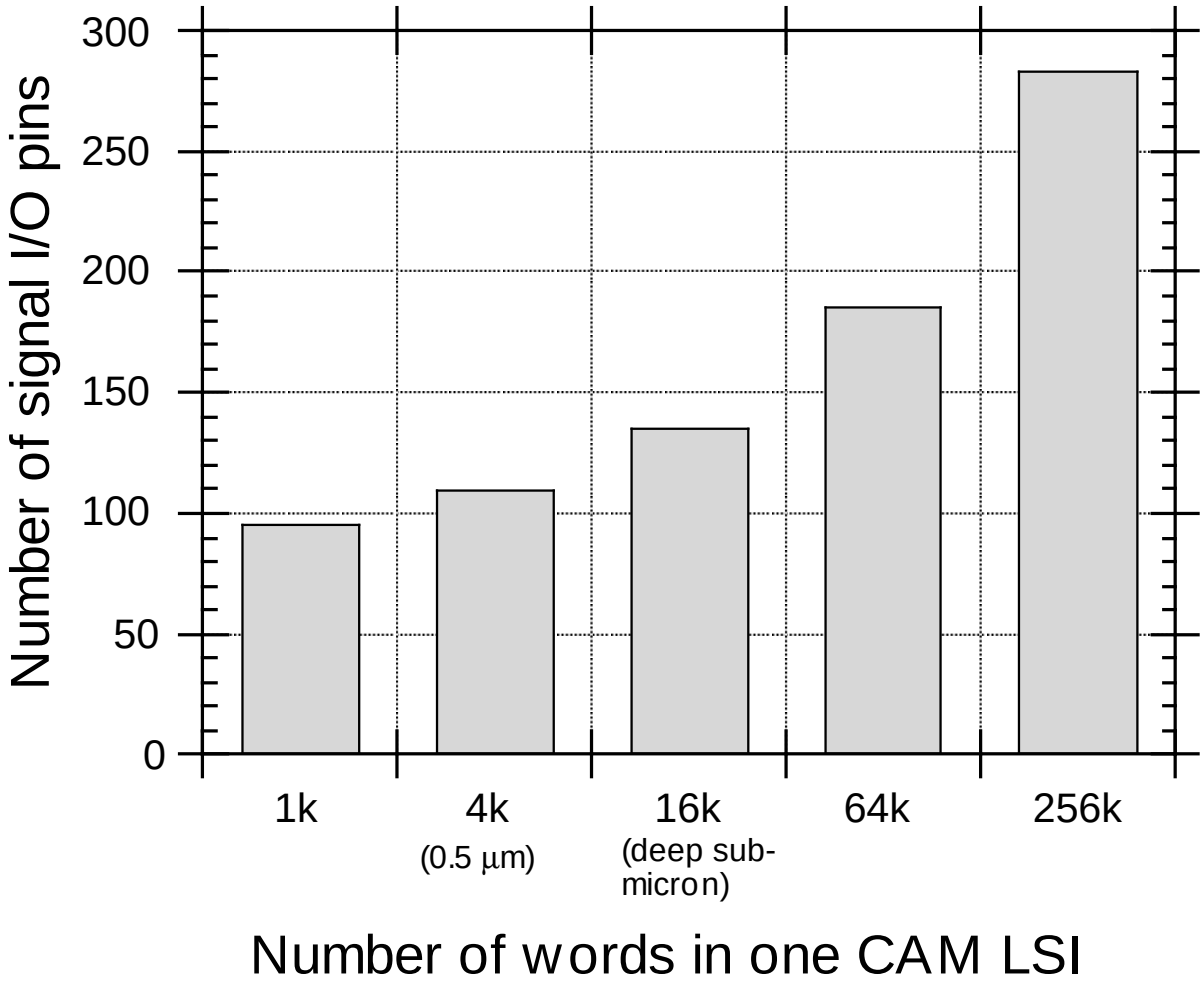


Figure 2.12: Number of signal I/O pins.

Each CAM consists of CAM blocks with 1 k words. Since 64 bits are assigned to one

word, the 4 k and 16 k CAM LSIs in Fig. 2.12 correspond to the 0.5-μm CAM with 256 k bits and the deep sub-micron CAM with 1 M bits in table 2.1, respectively.

The number of signal I/O pins of the 4 k and 16 k CAM LSIs is less than 150. Considering today's LSI packaging technology, this number of pins is easy to implement. Furthermore, although the increase rate of the Field Data I/O and Hit Data I/O pins is proportional to the number of words, the increase rate of the Address I/O pins is logarithmic and the number of Data I/O and Instruction Input pins is fixed. Therefore, the increase rate of I/O pins is not so high. Larger CAM LSIs adopting future process technology can also be implemented without causing I/O bottlenecks.

2.5.2 Processing performance evaluations

CA processing performance and data loading and retrieval processing performance are discussed in this section. In the evaluation, the following CAM² was assumed:

- 512 × 512 CA cells (16 CAM LSIs consisting of sixteen 1 k-word CAM blocks)
- m × n = 1024 (corresponding to the number of the words of one CAM block)
- 40-MHz system clock

For the evaluation, the functional hardware model of CAM² based on Verilog-HDL [55] was developed.

CA processing performance

The CA processing performance of CAM² is shown in Fig. 2.13. In the evaluation, the four-neighborhood CA processing model whose transition rule is based on mathematical morphology [12] is used. The processing method of morphology will be explained in detail in Chapter 5.

When the transfer bit width is small, the intra-CAM transfer becomes dominant. Therefore, the bigger m is, the shorter the transfer cycle becomes, as shown in Fig. 2.13. On the other hand, when the transfer bit width is big, the inter-CAM transfer becomes

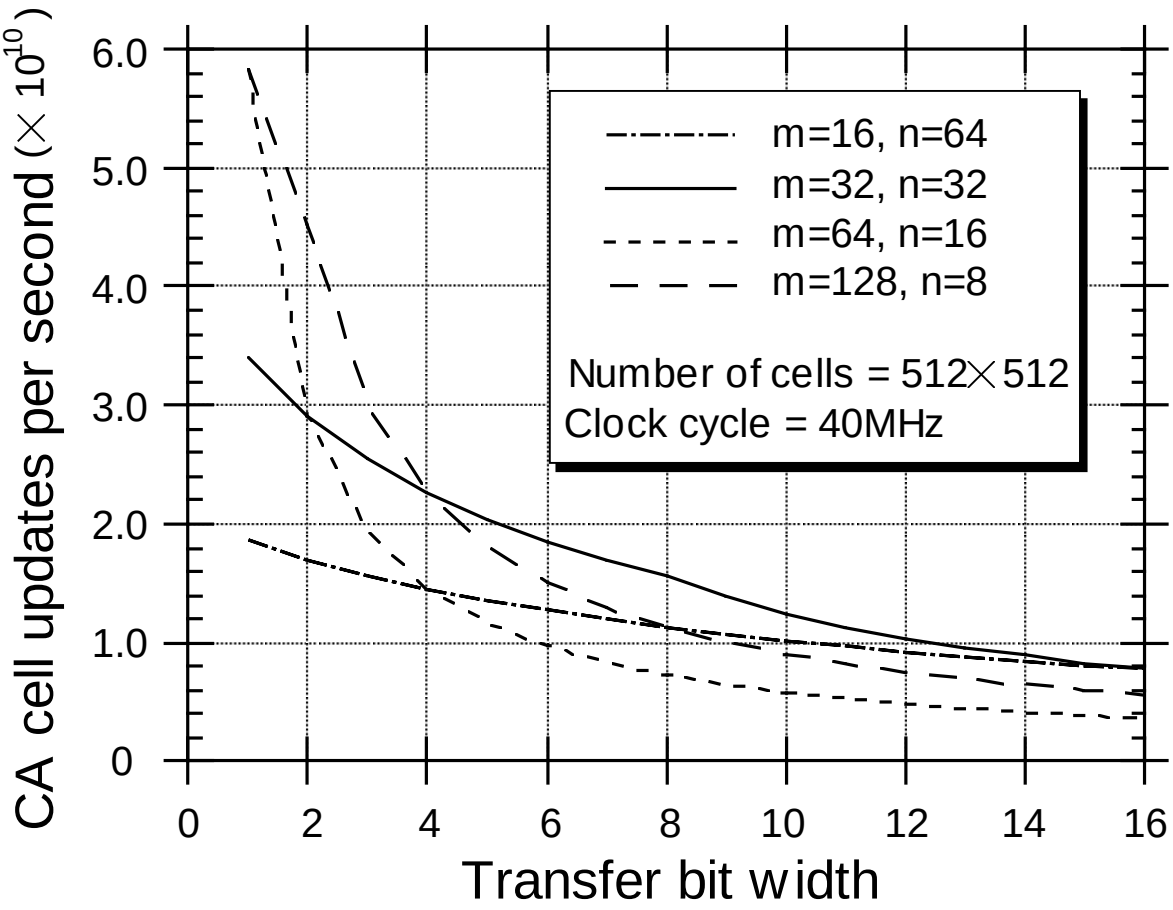


Figure 2.13: CA processing performance.

dominant. Therefore, the smaller m is, the longer the transfer cycle becomes. Thus, proper m must be chosen according to the transfer bit width to achieve higher levels of performance.

Upon choosing the proper m, 9 billion - 58 billion CA cell updates per second can be achieved. In terms of image processing, this means that more than a thousand types of CA-based image processing can be executed to the all 512×512 pixels at video rates (33 msec). Therefore, CAM² has a large potentiality for real-time pixel-level image processing that requires CA processing with various transition rules.

Data loading and retrieval performance

Figures 2.14 and 2.15 shows the performance of the conventional and proposed methods for data loading and retrieval, respectively.

The processing time of the conventional methods is proportional to the number of words. For example, the total processing time of CAM² with 256 k words for both data loading and retrieval is about 13 msec, which is too large considering that for real-time applications the processing must be performed at video rates (33 msec).

On the other hand, when the number of words is small, the processing time of the proposed methods is not so different because of the overhead for the inter-field and inter-word transfer. But when the number of words is big, our methods are about 8 times faster than the conventional methods. Since the total processing time of CAM² with 256 k words for both data loading and retrieval is about 1.6 msec, more than 30 msec can be used for CA processing for real-time applications.

For the retrieval, a multiple-response resolution (MRR) [22] technique is also available. Using MRR, only hit words can be retrieved. The number of cycles needed for the MRR is $2s - 1$, where s is the number of hit words. So, if the hit word ratio is less than 6.25%, the processing time of MRR is shorter than that of the proposed method.

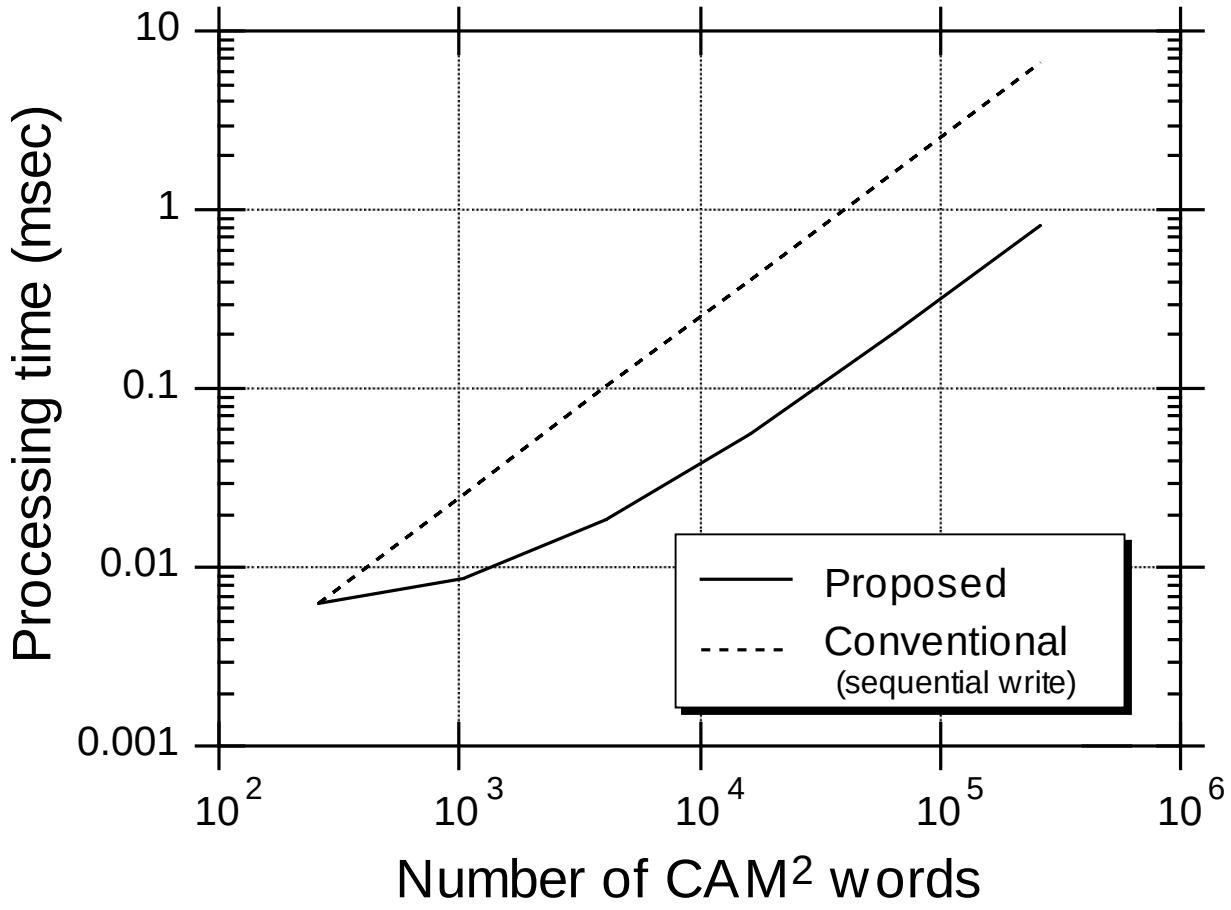


Figure 2.14: Data loading performance.

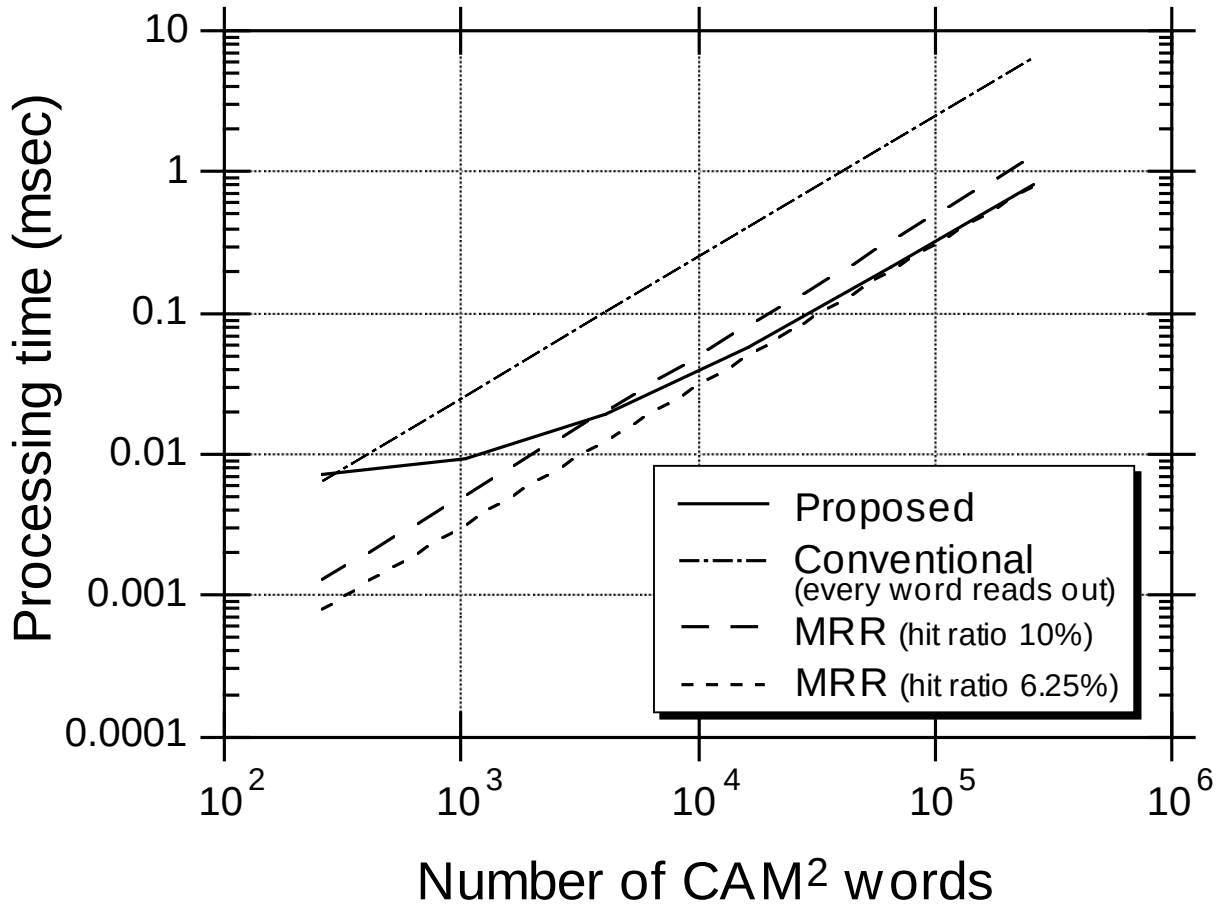


Figure 2.15: Data retrieving performance.

2.6 Conclusion

In this chapter, I proposed a highly-parallel two-dimensional cellular automata architecture called CAM^2. CAM^2 can attain pixel-order parallelism on a single board because it is composed of a CAM, which makes it possible to embed enormous numbers of PEs, corresponding to CA cells, onto one VLSI chip. Multiple-zigzag mapping of a CA cell to a PE combined with dedicated CAM functions enables high-performance CA processing. This mapping realizes high throughput CA-value transfer between neighbor CA cells, even though a CAM has only a one-dimensional data transfer path. Parallel loading and partial word retrieval techniques enable high throughput data transfer between CAM^2 and the outside image buffer.

The performance evaluation results show that 256 k CA cells, which correspond to a 512 $\times$ 512 picture, can be processed by a CAM^2 on a single board using deep sub-micron process technology. The processing speed is more than 10 billion CA cell updates per second under a four-neighbor condition. This means that more than a thousand CA-based image processing operations can be done on a 512 $\times$ 512 pixel image at video rates (33 msec).

CAM^2 enables us to realize a board-sized two-dimensional CA that can process various real-world images of several hundred thousand pixels at video rates. Thus, CAM^2 will widen the potentiality of a CA and make a significant contribution to the development of compact and high-performance information systems.

Chapter 3

1-Mb CAM LSI and CAM^2 board

3.1 Introduction

In this chapter, I describe a fully-parallel 1-Mb CAM LSI with dedicated functions for CA processing and a proof-of-concept prototype CAM^2 system (PC board) using this LSI from the viewpoint of design and implementation. The design is based on conclusions drawn from the architectural research presented in the preceding chapter. The LSI and the board demonstrate that economically feasible compact, high-performance, and flexible CAM^2 can be actually obtained with current technology.

There are two major problems as regards developing a large capacity CAM chip based on state-of-the art LSI technology (0.25 μm process technology was used in this research). These problems are 1) how to satisfy a number of physical constraints, such as wiring delay, power consumption and packaging, and 2) how to design and confirm the affectiveness of the chip for rapid implementation. I propose a scheme that combines one-dimensional (intra-block) and two-dimensional (inter-block) physical structures and comb data and clock distribution to solve the former problem. For the latter problem, I devised both manual and synthetic design strategies. I also propose new CAM functions for faster multiple-bit logical and arithmetic operations and global data operations (these global operations are beyond the CA concept but they are required to perform certain kinds of image processing algorithms). Moreover, this chapter presents the results of various estimations related to such factors as operating frequency, power consumption and chip

size, based on fabricated chips.

I expect the full potential of CAM^2 to be demonstrated when a faster and higher capacity system is realized by streching current LSI and board technologies to or even beyond their current limit. However, in reality an actual LSI and board must meet the requirements of a reasonable design cost and time Thus, before starting LSI and board design, I clarified the design constraints based on the available (1997) LSI and board technologies. Below I have summarized the design constraints imposed on the design process.

LSI chip

- 0.25 μm, five aluminum layer, full-custom CMOS fabrication process
- No more than 200 signal I/O pins per chip (allows use of regular pin pitch, 0.5 mm, QFP package)
- Less than 2 watts of maximum power dissipation per chip (allows use of plastic mold package that can be air-cooled)
- Less than 17.0 $\times$ 17.0 mm die size (allows use of 28 mm square QFP package)

PCI board

- 312 $\times$ 107 mm full-size PCI board (allows use of conventional PC as host computer)
- Less than 20 LSI components
- Less than 40 MHz system clock (allows easy development of control logic on FPGAs)
- Utilization of off-the-shelf components, except for fabricated CAM LSI

Based on these constraints, I selected 1-Mb CAM LSI as a target. Since it has 16 k words, or processing elements (PEs), which can process 128 $\times$ 128 pixels in parallel, a board-sized pixel-parallel image processing system can be implemented using several

chips. Moreover, the targeted board consists of a highly-parallel PE array (which is a two-dimensional 2 × 2 array of 1-Mb CAM LSIs), FPGAs and some memory. It can process 256 × 256 pixels in parallel. In addition, PCI bus and NTSC video interfaces are also embedded in this board. This means that a compact image-processing platform can be built simply by connecting the board to a personal computer and a video camera.

This chapter is organized as follows. Section 3.2 presents methods for designing a large capacity CAM. This is followed, in Section 3.3, by a description of CAM functions for performing various image-understanding algorithms. After discussing the design and verification procedure for the rapid implementation of this chip in Section 3.4, I present the specifications and processing performance of a fabricated chip and a prototype CAM^2 board using the CAM LSIs in Sections 3.5 and 3.6, respectively.

3.2 CAM LSI design

3.2.1 Chip level design

Figure 3.1 is a block diagram of the CAM chip. In fabricating a high-capacity LSI with pixel-order PEs, it is especially important to make full use of the CAM's memory-based one-dimensional physical structure. However, connection networks that provide high-throughput data transfer are also important. To satisfy both requirements, the LSI uses a scheme combining a one-dimensional physical structure within blocks and a two-dimensional physical structure between blocks. The LSI is thus divided into 32 CAM blocks (U0-U15 and D0-D15), each with 512 words. Horizontally and vertically adjacent CAM blocks are connected by an 8-bit Hbus and a 1-bit Vbus, respectively. Since both the Hbus and Vbus are also distributed outside the chip in the same manner, a larger CAM array can be obtained just by using several chips and connecting these buses to each other in a mesh.

A chip controller (Chip CNT) controls the whole chip. It distributes various global signal data, such as control signals (decoded instructions), data and address, to all CAM blocks through a main bus. It also calculates a single-hit flag (only one word is selected

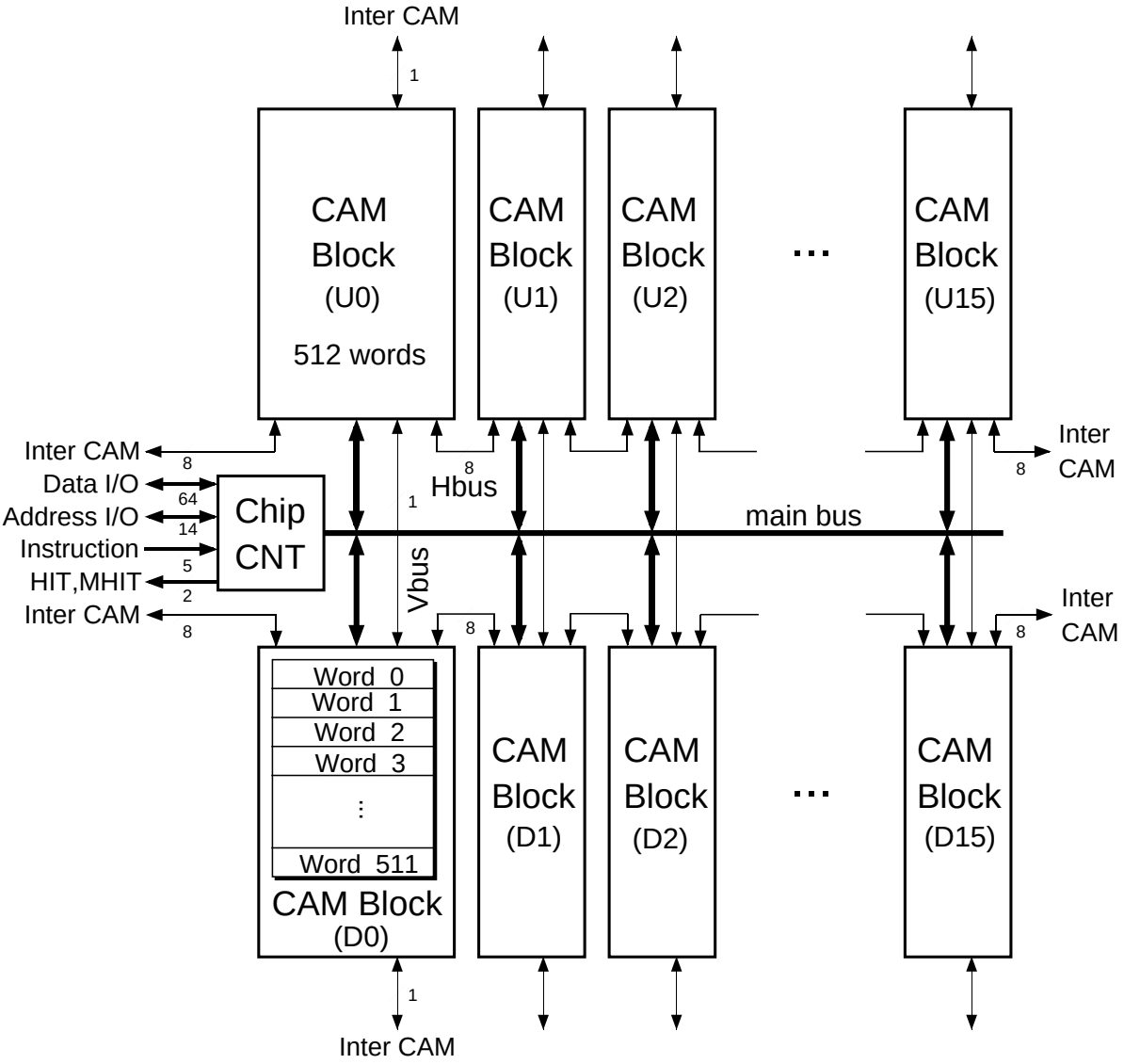


Figure 3.1: Block diagram of the 1-Mb CAM.

or not selected) and a multiple-hit flag (plural words are selected or not selected) in a search operation by compiling hit flags and output address data from all CAM blocks, and outputs them through the HIT and MHIT port, respectively.

Since this CAM does not have a complete two-dimensional structure, it cannot handle two-dimensional pixel data as it is. To assign two-dimensional pixels to it, a procedure called multiple-zigzag mapping is devised as shown in Section 2.2.3. I did not employ single-zigzag mapping because it takes too many cycles to transfer a value from one cell to a vertically adjacent neighbor. By assigning the words in each CAM block to an array of 6 $\times$ 64 pixels, one chip can logically handle 128 $\times$ 128 pixels in parallel, as shown in Figure 3.2.

3.2.2 CAM block and cell circuit level design

Figure 3.3 shows a block diagram of the CAM block. Each CAM block consists of a cell array block, a bit operation block, a word operation block and an interface block. The cell array block is composed of 512 word $\times$ 64-bit-writable and 18-bit-read-only blocks and an additional ROM block. Each cell in the writable block is composed of a latch and an Exclusive NOR circuit with 12 transistors as shown in Figure 3.3. The writable block can perform a maskable search, where search-result flag signals for each word are generated by using the output signal of each Exclusive NOR circuit and are sent to hit-flag registers in the word operation block in a parallel fashion. It is performed by detecting whether or not a precharged match line, ML, has discharged through the cell. Moreover, a partial & parallel write can also be performed, where the data are written into specific bit positions of multiple words. For a write operation, writing into all the cells connected to the same bit line is prohibited if the signal PWj is set to a low level. Input capacitance of some transfer gates, such as one connected by KDj line, changes depending on stored data. So, the size of devices that drive such cells was carefully decided through worst case simulation. During read and write operations, the electrical path to Vdd is cut off by Transistor 1. So, the bit line can be driven even if multiple cells are active at the same

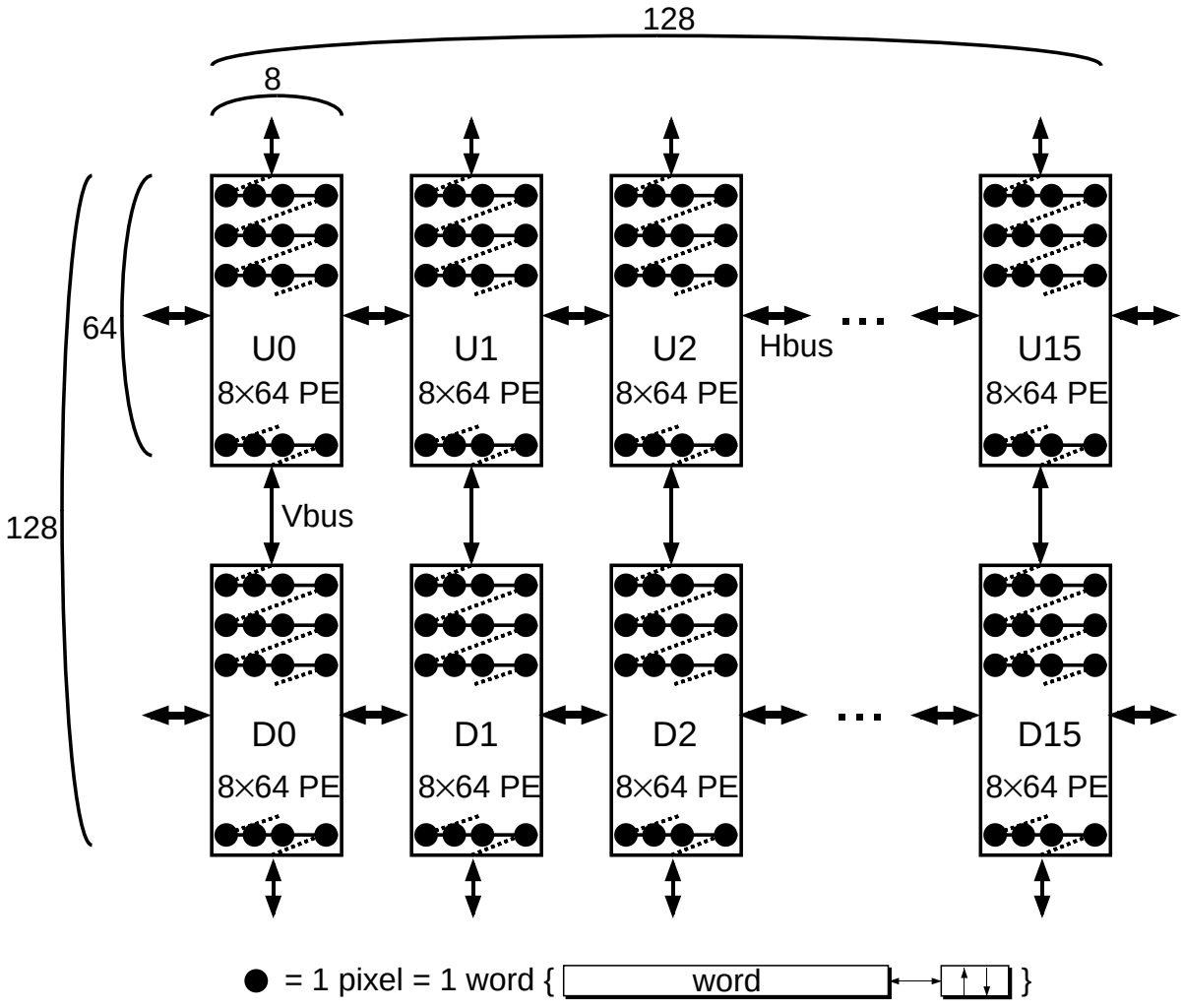


Figure 3.2: Logical configuration (multiple zigzag mapping).

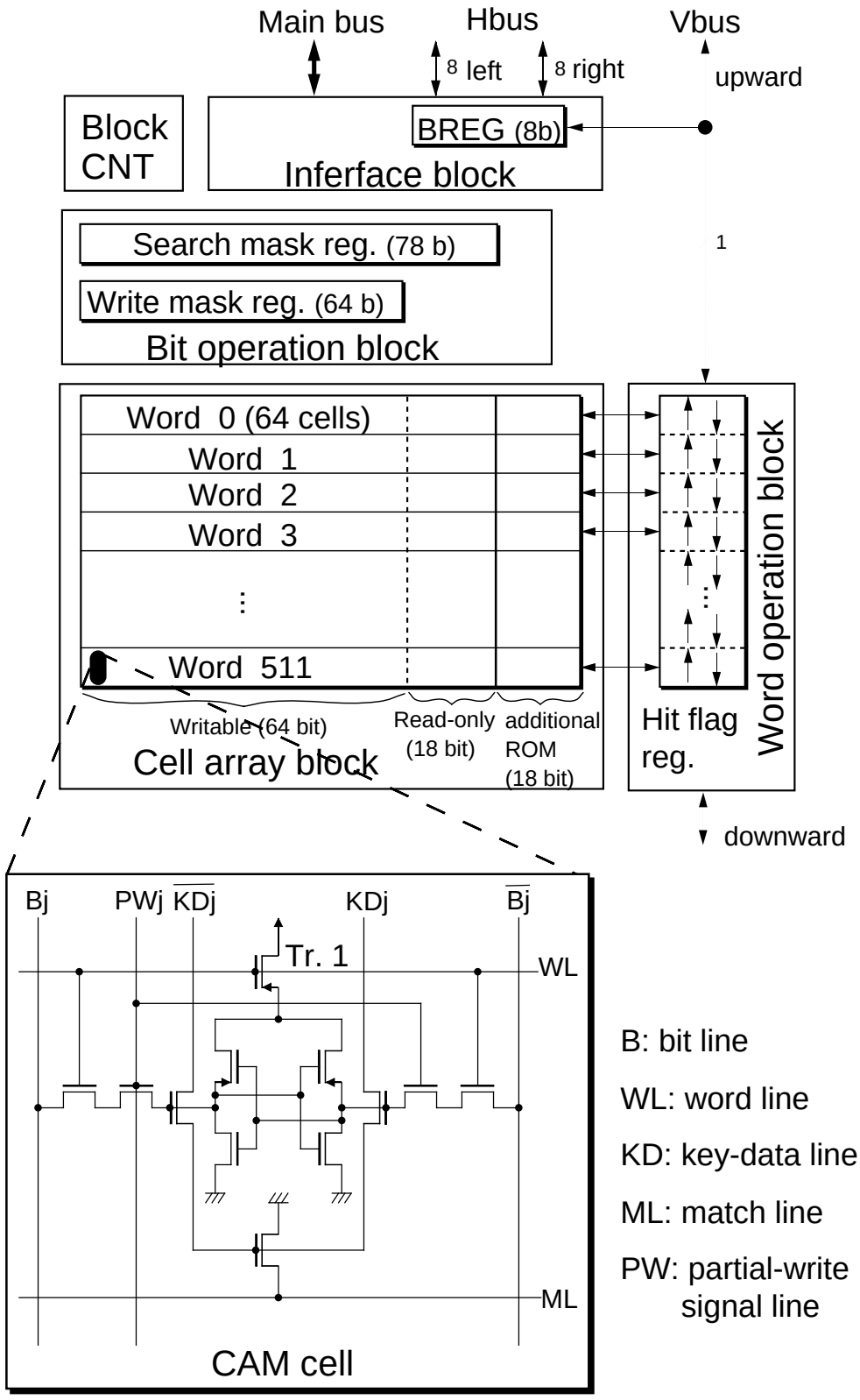


Figure 3.3: CAM block and cell circuits.

time. However, since the read operation to writable block is only allowed in single hit case and contents of read-out data are not guaranteed in multi hit case, hit flags must be checked before executing the read operation.

The read-only block is composed of ROM cells and a maskable address decoder, where word-address data are stored. The read-only block can also perform a maskable search for the fixed address data. For example, plural words can be address by masking them. The complementary address data are stored in the additional ROM block. These ROM data are used for generating single- and multiple-hit flags in word operation block. The word operation block generates single- and multiple-hit flags using read-out data from the ROM in the cell array block.

The ROM outputs "0" when no word-line is driven. When a single word-line is driven, the stored data are output. If plural word-lines are driven, logical OR of stored data of same bit-line is output. Therefore, by examining the output address-data and their complement data, single- and multiple-hit flags can be generated. The flag generation logic is expressed as,

$$HIT = \wedge i(Ai \oplus Bi) \tag{3.1}$$

and

$$MHIT = (\vee i(Ai) + \vee i(Bi)) \wedge (\neg HIT) \tag{3.2}$$

where HIT is a single-hit flag, MHIT is a multiple-hit flag, $\wedge$ is AND, $\vee$ is OR, $\oplus$ is exclusive-OR, $\neg$ is negation, and Ai and Bi are word-address data and their complement data, respectively. Since ROM is small and regular-structured, single- and multiple-hit flags generation function can be implemented by extremely small amount of hardware. In the word operation block, a hit-flag register is also provided for each word. The accumulation of the results of the bit-serial search for parallel processing is carried out by this register.

The bit operation block controls the bit position for the search and the parallel write. It is composed of a 78-bit search mask register which specifies search bit position and a 64-bit write mask register which specifies write bit position. The interface block switches the path from/to the Chip CNT and the path from/to vertically adjacent blocks. It has a pipeline register (SREG) for data transfer between vertically adjacent blocks. An SREG also acts as a one-bit counter for hit flags.

3.2.3 Data and clock distribution

Another important feature of the configuration is global signal distribution, whereby data, address and clock signals and such are distributed to all 32 blocks. With this scheme, the wiring delay is the most serious problem because, for a 0.25-μm process, it increases exponentially with wire length. Moreover, in implementing a large-capacity CAM, I have to try to keep the peak power consumption down. To solve these problems, a comb-distribution scheme is used, in which the data and clock are distributed in the same way, as shown in Figure 3.4.

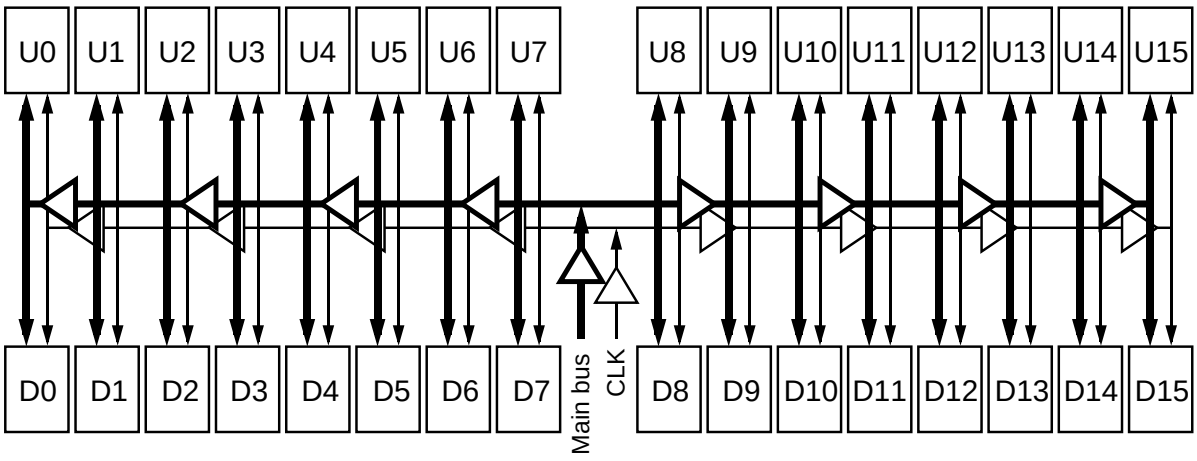


Figure 3.4: Data and clock distribution scheme.

Buffers inserted every five or six fanouts reduce the wiring delay. The different numbers of buffer stages produce an intentional skew, which lowers the peak power. But, since the skews between horizontally and vertically adjacent blocks are both less than 0.5 ns, data

can be transferred without causing timing problems.

3.3 Instruction design

3.3.1 Data transfer and update operations

CA processing is carried out by the iterative operations of CA-value transfer and update as shown in Section 2.3. For data transfer, this CAM LSI supports upward/downward hit-flag shift and block read/write. For the shift, the hit-flag registers have a bi-directional shift mode. Since the hit-flag registers between vertically adjacent blocks are connected through a Vbus, the vertically adjacent blocks are considered to be one block in the shift mode. With block read/write, optional 8-bit data can be transferred to an optional word of the horizontally adjacent CAM block through an Hbus. An SREG in the interface block is used as a pipeline register for the transfer. As shown in Figure 3.5, data is transferred within blocks and between blocks.

Intra-block transfer deals with inner words and is carried out by the iteration of the maskable search, the hit-flag shift (1 and 8 bit shift to the left/right PE and upper/lower PE, respectively), and the parallel write. Thus, hit-flag registers can be utilized as a one-bit data transfer path. On the other hand, the horizontal data transfer for boundary words (the white circles in Figure 3.5) cannot be performed within a block. Inter-block transfer deals with this and is carried out by means of the block read and write.

Two functions are absolutely essential for the update operation: a maskable OR search, and partial and parallel writes. For the search, the results are accumulated in hit-flag registers by means of OR logic. For the writes, the data are written into specific bit positions of multiple words for which the value of the hit-flag register is 1. Through the iteration of these two operations, SIMD-type update operations can be carried out in a bit-serial, word-parallel manner.

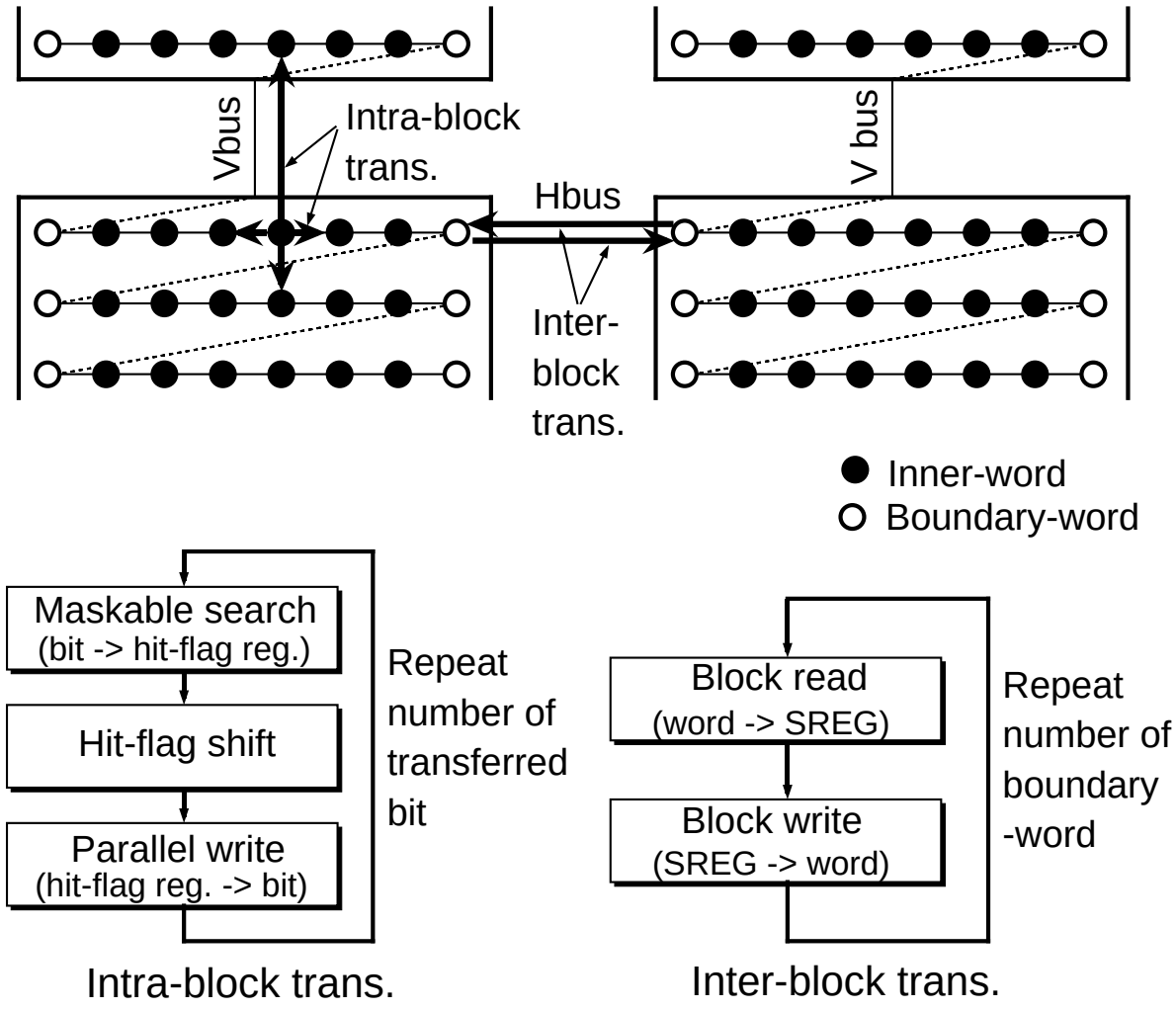


Figure 3.5: Data transfer method.

3.3.2 Multi-bit arithmetic operations

For multiple-bit operations, the mask position must be changed according to the processed bit. I devised search or parallel write functions with an upward/downward mask shift to execute them faster. These functions can be implemented just by making minimum refinements; only replacing the search and write mask registers to shiftable ones. They do not affect CAM's high density structure at all.

Using these functions, the f-bit GREATER-THAN operation explained in Section 2.3 is executed in the following steps:

1. Set search mask.
2. Set write mask.
3. Maskable search for words whose Xi, Yi, F1, F2 is 0, 1, 1, 1.
4. Maskable OR search for words whose Xi, Yi, F1, F2 is 0, 1, 0, 0.
5. Parallel writing of 1, 0, 0 to Xi, F1, F2 of the hit words.
6. Maskable search for words whose Xi, Yi, F1, F2 is 1, 0, 1, 1.
7. Parallel writing of 1, 0, 1 to Xi, F1, F2 of the hit words.
8. Maskable search for words whose Xi, Yi, F1, F2 is 1, 0, 0, 0 and downward write mask shift.
9. Parallel writing of 0, 0, 0 to Xi, F1, F2 of the hit words and downward search mask shift.

Although the processing must be repeated from MSB (I = f) to LSB (I = 1), steps 1 and 2 are cut from the second iteration by using the proposed functions. The number of cycles for the f-bit GREATER-THAN operation is

$$C_{greater-than} = 7 \times f + 2. \tag{3.3}$$

So, the proposed functions bring the processing time for 8 bit GREATER-THAN operation down to about 80% that of the conventional one explained in Section 2.3.

3.3.3 Global data operations

To cover various image-understanding algorithms, not only local operations but also global data handling is indispensable. The global operation is beyond CA concept but is indispensable to perform a certain kinds of algorithms. For example, the period of DTCNN dynamics changes according to the geometric features of the input image. So, convergence assessment is required to eliminate redundant transitions. Moreover, the pattern spectrum processing based on morphological operations, which will be explained in detail in Section 5.4, is very useful for getting information on the global features of target objects. The processing, however, requires area calculation, where pixel values stored in all PEs must be summed up. This CAM also implemented dedicated functions to efficiently carry out such global data operations.

As examples of global data operations, convergence assessment and area calculation are shown here. To assess convergence DTCNN defined by Equations (4.1) and (4.2), which will be explained in Chapter 4, I must examine whether all $y(k)$ values are equal to $y(k-1)$ or not. The assessment can be executed efficiently by using hit flags (HIT and MHIT). This CAM can perform it in only several cycles as shown in the following steps:

1. Set search mask.

2. Set write mask.

3. Maskable search for words whose $y(k)$ is 1 and $y(k-1)$ is -1.

4. Maskable OR search for words whose $y(k)$ is -1 and $y(k-1)$ is 1.

5. Convergence assessment. (If "HIT OR MHIT $= 0$," then convergence occurs. Otherwise repeat new transition.)

Figure 3.6 shows the processing steps for area calculation, where it is assumed that a search of pixel values has been carried out and the results have been stored in hit-flag registers.

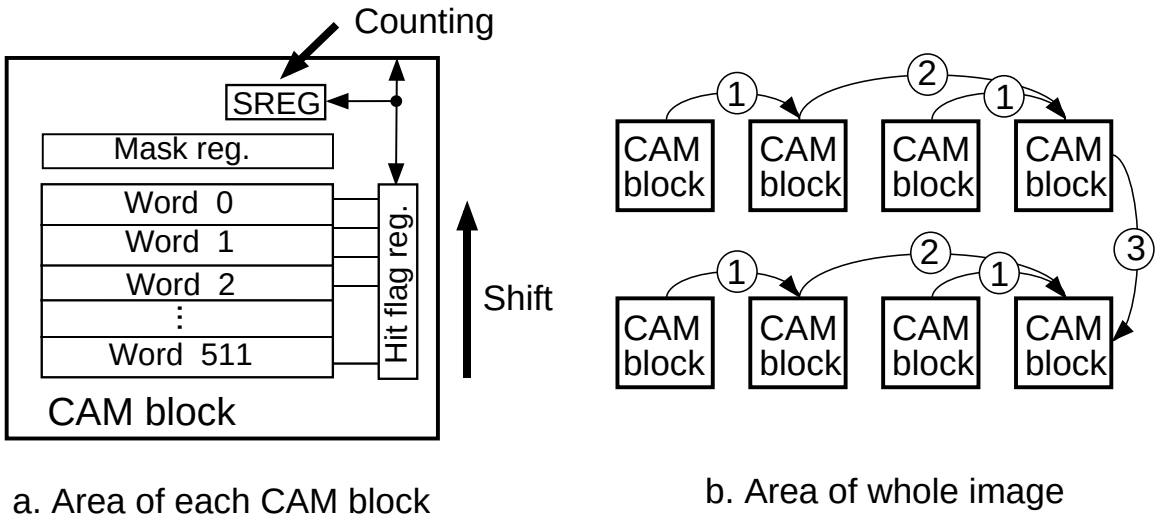


Figure 3.6: Area calculation. Area of (a) each CAM block and (b) whole image.

First, each hit-flag register is shifted repeatedly, and the carries are accumulated in a SREG that functions as a counter. This yields the area of each CAM block. Next, iterative data transfer through the V- and H-buses and addition yields the areas of all the CAM blocks. This operation moves through the area data in a tree-like fashion, and outputs the area of the whole image. The first step is carried out in parallel in each block. Moreover, the number of iterations of the next step varies logarithmically with the number of blocks. So, an area calculation takes only a short time.

3.3.4 Instruction set

The instruction set contains 32 instructions classified into five modes: search, read, write, data transfer and the rest, as shown in Table 3.1. Each operation takes just one cycle. Two-dimensional pixel-parallel processing is carried out through a combination of these instructions. It is upper compatible to the 336 k-bit CAM LSI [48] that we previously developed. The 1-Mb CAM also has various advanced functions of the 336 k-bit CAM,

Table 3.1: Instruction set of CAM LSI.

Mode	Operations
Search (5 instructions)	◇ fully-parallel maskable search ◇ maskable OR/AND search ◇ maskable search with write mask shift
Read (4 instructions)	◇ data read using address the same as RAM ◇ hit-flag, true and complement address read ◇ address & data read at single hit
Write (7 instructions)	◇ data write using address the same as RAM ◇ parallel write into matched words ◇ parallel write with search mask shift
Data Transfer (9 instructions)	◇ hit-flag shift up/down ◇ block data read/write ◇ block data read/write with hit-flag shift
Clear & Set & Count (6 instructions)	◇ hit-flag/counter clear ◇ search/write mask set ◇ hit-flag count
NOP	

such as multiple-response-resolution (MRR), where only hit words can be retrieved one by one. So, it can also efficiently handle not only pixel-parallel processing but also various image processing like Hough transform [24] – [27] and 3D-feature extraction [28].

3.4 Design and verification procedure

Figure 3.7 shows the design and verification procedure for this CAM. To speed up chip development while achieving a high-density structure, both manual and synthetic design strategies are employed. According to this, CAM cells and the word operation block, which contain a large number of transistors that need to be tightly packed, are designed with a macrocell entry tool. Since the other peripheral parts, like the controller and the bit operation block, have fewer transistors but are very complicated, they are designed by describing their behavior in a hardware description language. Mask data is obtained by the repetition of layout and timing analysis to a chip netlist, which is a combination of macrocell data and synthesized logic cell data. The comb data and clock distribution described in Section 3.2 is implemented by simply designating the driver position.

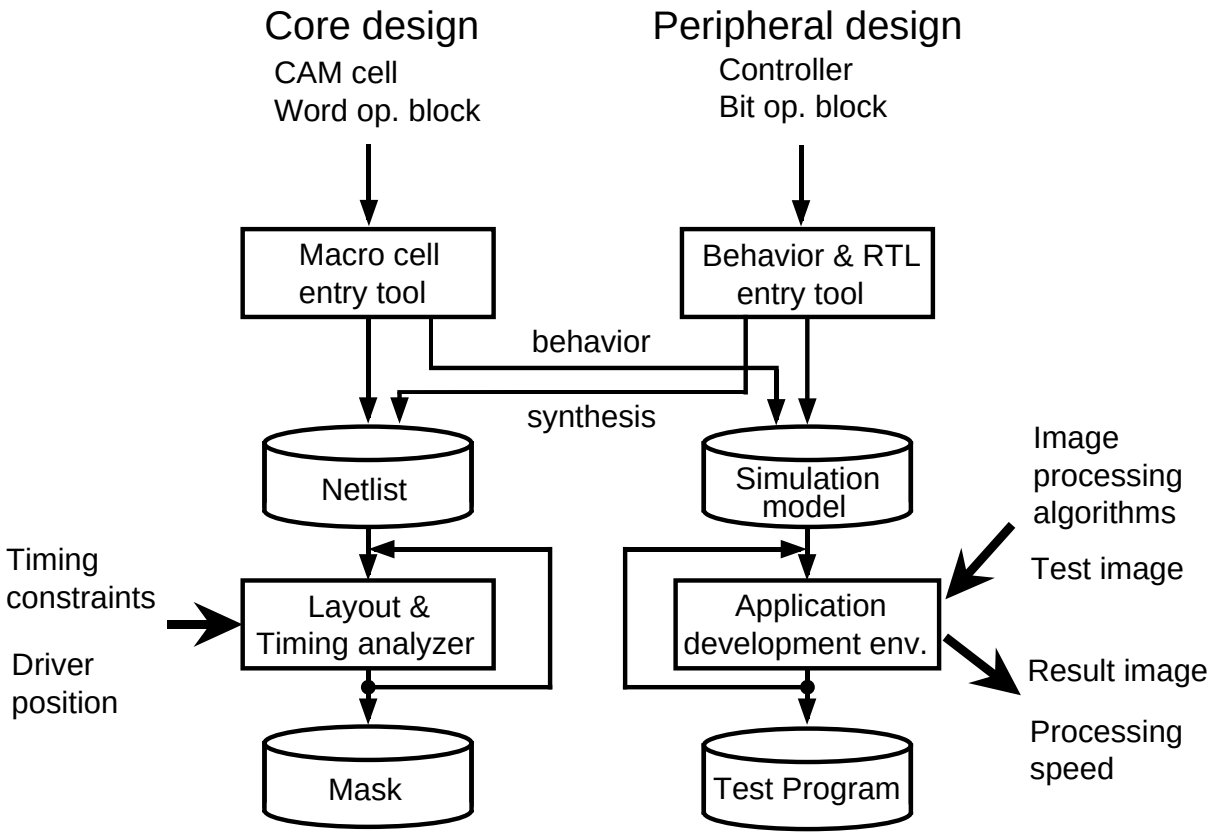


Figure 3.7: Design and verification flow.

For verification, I created an application development environment. It acts as a behavior simulator for the 1-Mb CAM and reports various simulation results, like output image and processing speed, for various image processing algorithms; This environment saves a great deal of verification effort. It also automatically generates a test program for test equipment. The application development environment will be explained in detail in Section 4.4.

3.5 Fabricated LSI and its specifications

The CAM LSI was fabricated using 0.25-μm full-custom CMOS technology with five aluminum layers. A micrograph of the chip is shown in Figure 3.8. One CAM cell occupies 7.7 $\times$ 11.9 μm. A total of 15.5 million transistors have been integrated into a 16.1 $\times$ 17.0 mm chip. There are 32 physical CAM blocks composed of 512 words. Each word functions as a processing element and handles one pixel. Since all of the peripheral logic (e.g. controller) and global signal wiring (e.g. data and address) are gathered in a center vertical part of the chip, the whole chip was efficiently implemented without waste area.

Figure 3.9 shows a Shmoo plot, which indicates that this LSI is capable of operating at 56 MHz and 2.5 V. Table 3.2 of the chip specifications summarizes the design and implementation results. All data are based on measured silicon.

At 40 MHz, the maximum power dissipation is 2.3 W. Since the power of the CAM changes dramatically depending on the type of data being processed and the kinds of instructions, the typical power dissipation should be lower than that. For example, for edge detection it is 0.25 W. Table 3.3 shows the processing performance at 40 MHz.

Typical arithmetic and data transfer operations take from 0.025 to 5 μsec., which is equivalent to a performance of 3 to 640 GOPS per chip. And typical pixel-parallel operations, such as edge detection and hole filling, take less than 20 μsec. This means that more than a thousand pixel-parallel update operations can be done at video rates. So, this LSI has great potential for real-time image processing by means of a combination

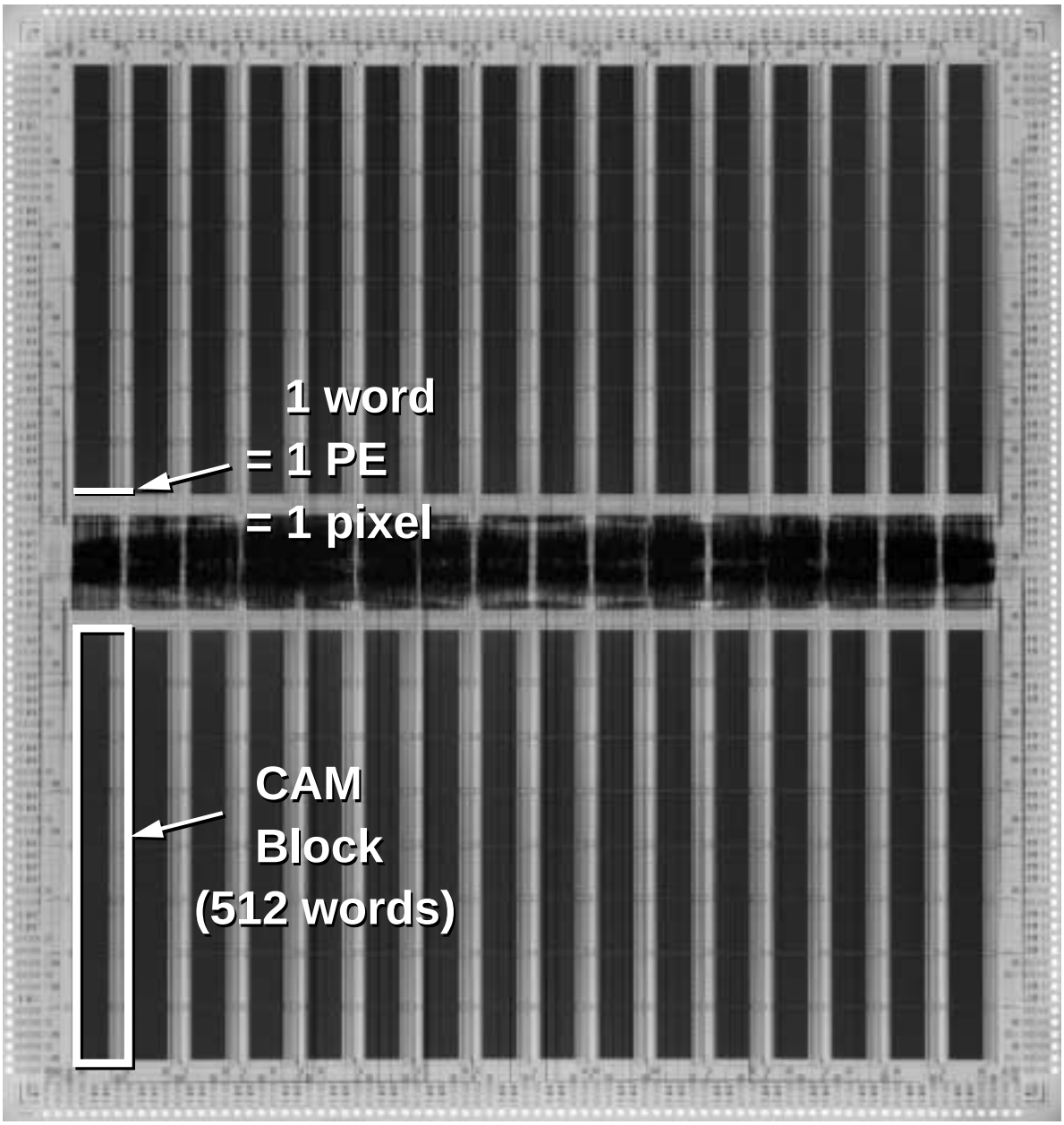


Figure 3.8: Chip microphotograph.

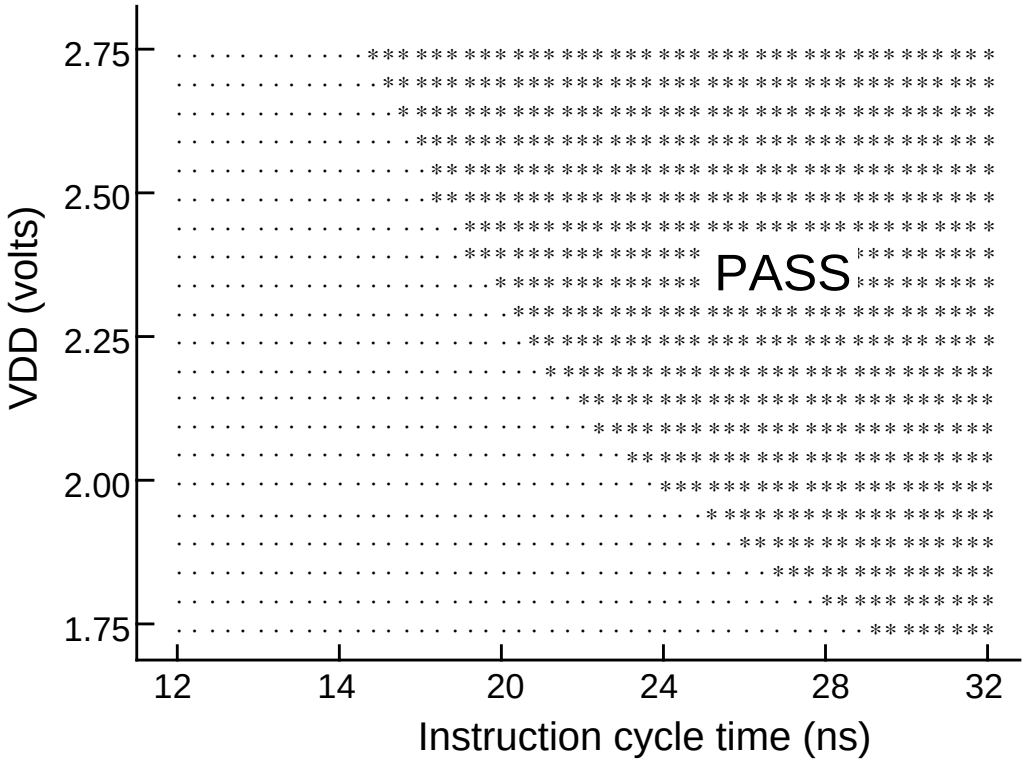


Figure 3.9: Shmoo plot.

Table 3.2: Specification of CAM LSI.

Configuration	512 words × (64 + 18) bits × 32 blocks
Instruction set	32
Operation frequency	56 MHz (typical
Supply voltage	2.5 V (3.3V for I/O)
Power dissipation	2.3W at 40 MHz (maximum)
	0.25W at 40 MHz (edge detection)
Number of pins	155
Package	208-pin P-QFP
LSI process technology	0.25 μm CMOS
	(five aluminum layers)
Number of transistors	15.5 million
Chip size	16.1 × 17.0 mm

Table 3.3: Processing performance at 40 MHz.

64 bit search	0.025 μs (640.0 GOPS per chip)
16 bit constant data set	0.13 μs (180.0 GOPS per chip)
4 bit addition	0.85 μs (18.8 GOPS per chip)
8 bit addition	1.7 μs (9.7 GOPS per chip)
16 bit addition	3.3 μs (4.9 GOPS per chip)
8 bit Intra-word (PE) transfer	0.45 μs (35.6 GOPS per chip)
8 bit four-neighbor Inter-word (PE) transfer	4.7 μs (3.4 GOPS per chip)
Edge detection for 8 bit gray scale image	16.9 μs (morphology: dilation, rhombus) + 1.7 μs (subtraction)
Hole filling for binary image	14.1 μs per transition (Discrete-time CNN)

of various pixel-parallel operations.

3.6 A proof-of-concept prototype CAM^2 board

Based on the HiPIC concept, I designed and fabricated a proof-of-concept prototype CAM^2 PC board using fabricated CAM LSIs. Figure 3.10 shows a block diagram of the board.

The board consists of a highly-parallel PE or CA cell array, FPGAs and some on-board memory. The PE array is a two-dimensional array (2×2) of the 1-Mb CAM LSIs, and can process 256×256 pixels in parallel. The CAM LSI is scalable, which means that a larger image can be processed by increasing the number of chips. Furthermore, since this CAM LSI has very high levels of performance, the frame rate can be boosted to over 30 frames per second by using a faster sampling camera.

Two XC4036XLs (FPGA1 and FPGA2) and one Xilinx XC4010 (FPGA3) are employed as FPGAs. XC4000XL series high-performance, high-capacity FPGAs [56] provide the benefits of custom CMOS VLSI, while avoiding the initial cost, long development cy-

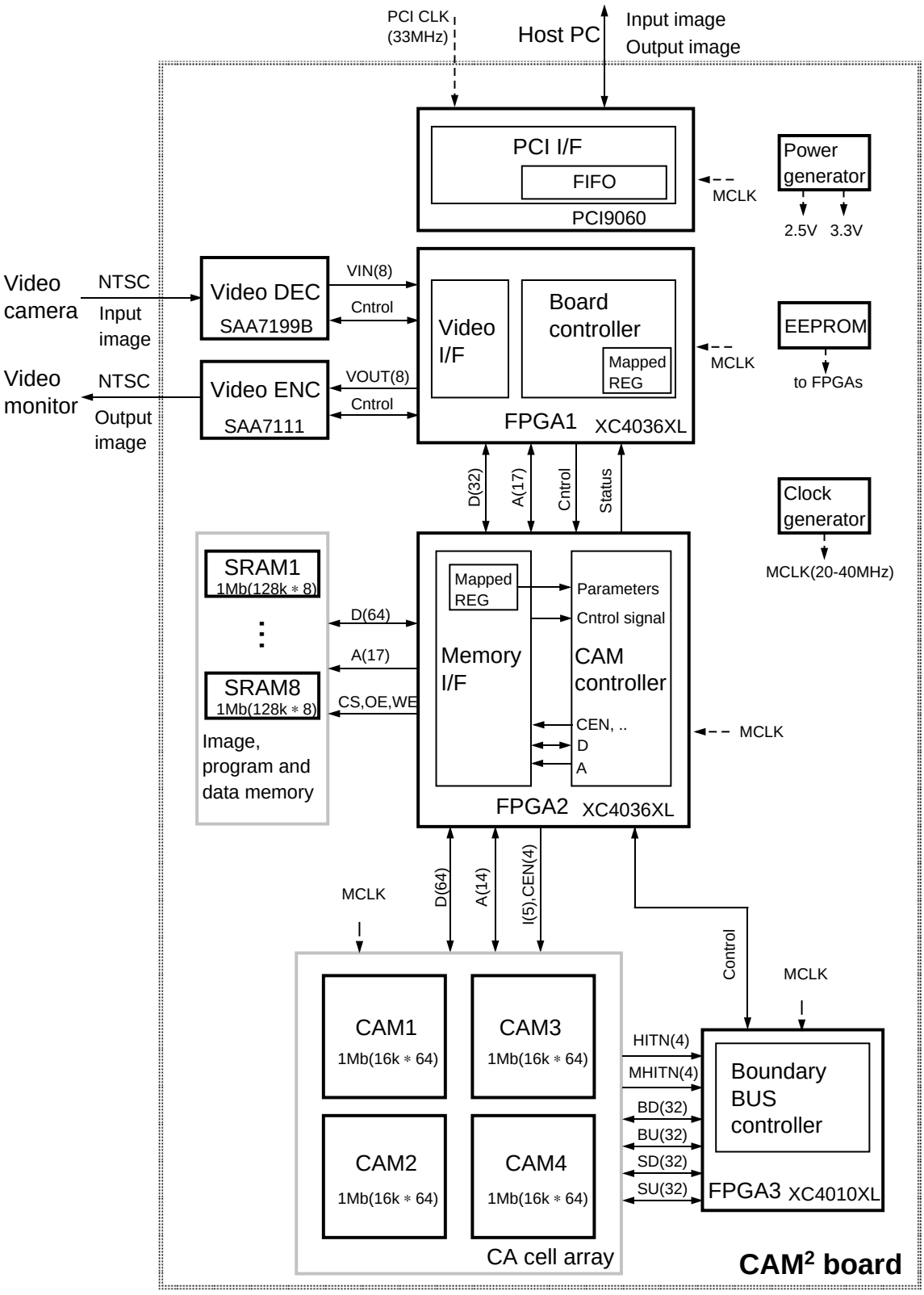


Figure 3.10: Block diagram of CAM^2 board with four CAM LSIs.

cle, and inherent risk involved in using a conventional masked gate array. These FPGAs combine architectural versatility, on-chip Select-RAM memory with edge-triggered and dual-port modes, increased speed, abundant routing resources, and software to achieve the fully automated implementation of complex, high-density, high-performance designs.

FPGA1 consists of a board controller and a video interface. The board controller controls the whole board, dealing mainly with various control signals from a host PC. The mapped registers on the controller can be directly accessed by the PC. The video interface transfers input image data captured by a video camera or stored on the host PC to the on-board memory. It also sends results of processed data stored in the on-board memory to a video monitor or to the PC. FPGA2 consists of a CAM controller and a memory interface. The CAM controller decodes microprograms stored in the on-board memory and generates command sequences for the CA cell array. Since an FPGA can easily generate various command sequences, it performs practical image processing by combining various types of pixel-parallel processing. The memory interface controls SRAM chips and arbitrates between various types of memory access from the CA cell array, the host PC and the video interface. FPGA3 controls the boundary bus of the CAM chips. Since it supplies optional data, the boundary values of the CA cell array can be adaptively modified.

The memory consists of eight asynchronous SRAMs and contains 8 M bits or 8×8 $(= 64) \times 128$ k bits. Figure 3.11 shows a memory address map. The memory is divided into three areas: image, program and data. To enable one memory port to be shared by various types of memory access, I employed a double buffer strategy, where two image frame data with a size of 512 pixels $\times$ 512 pixels $\times$ 8 bits (gray scale) are stored in the image area. According to this strategy, when image data is stored in or retrieved from frame A, the CA cell array processes the image stored in frame B. By changing a parameter, an optional 256×256 image for the CA cell array can be selected from an image of 512×512. In contrast, the CA cell array handles frame A image data when input or output images are stored in or retrieved from the frame B area.

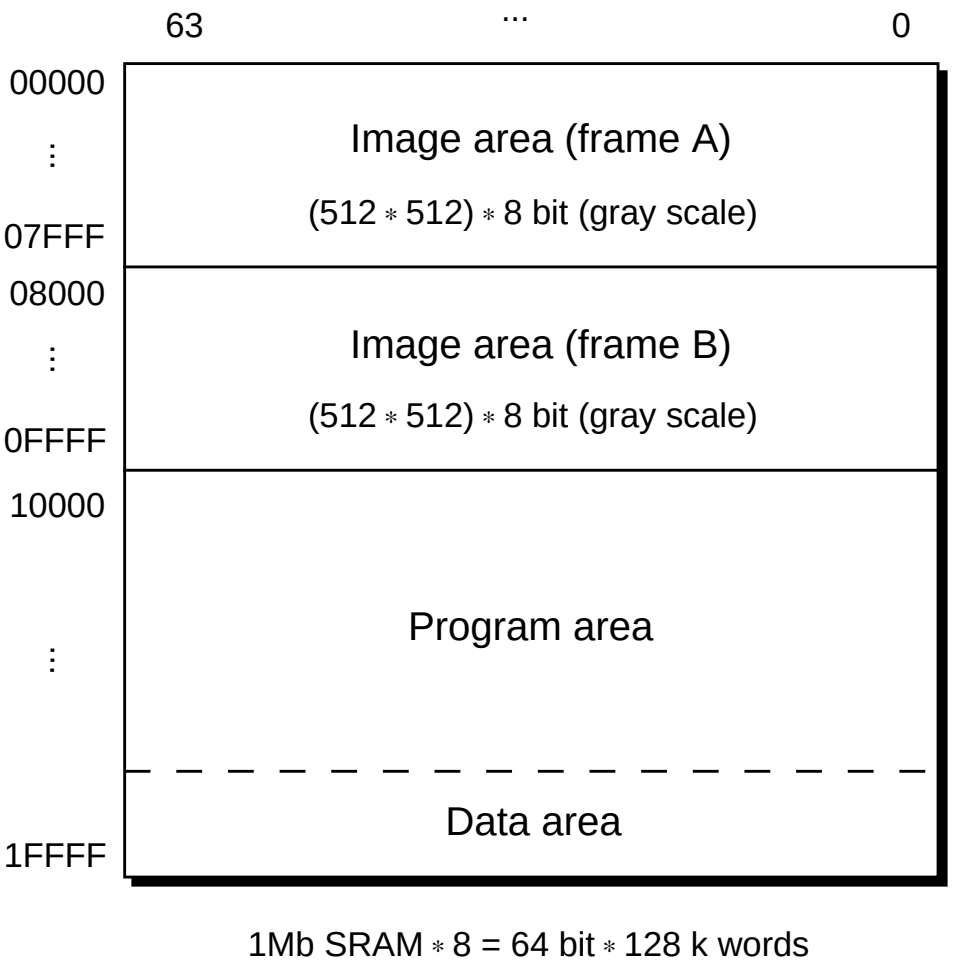
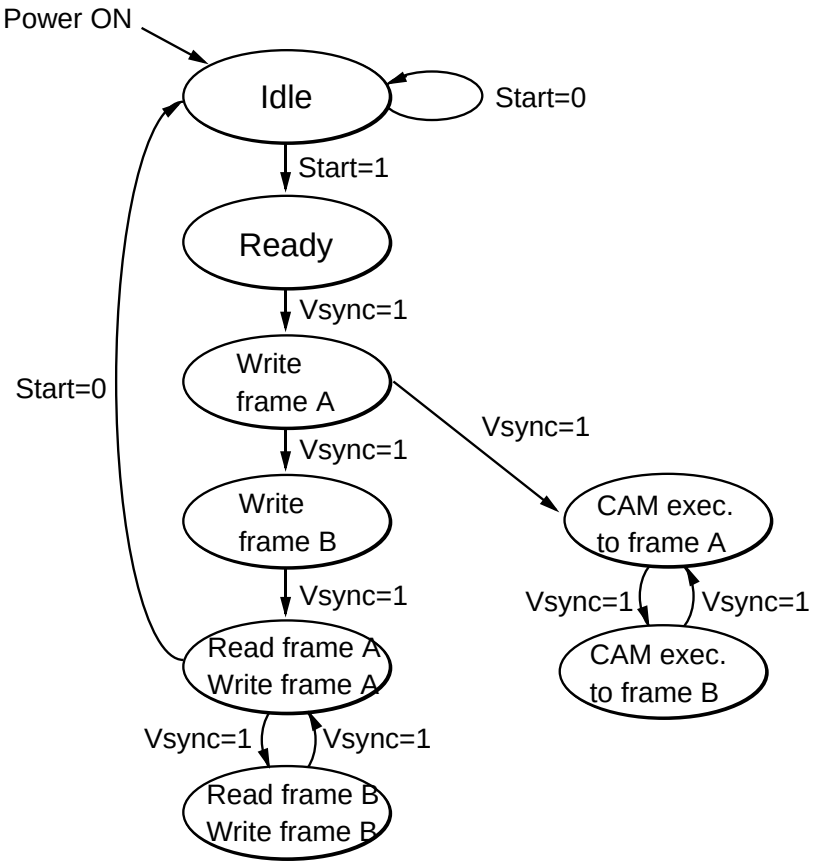


Figure 3.11: Memory address map of on-board SRAM.

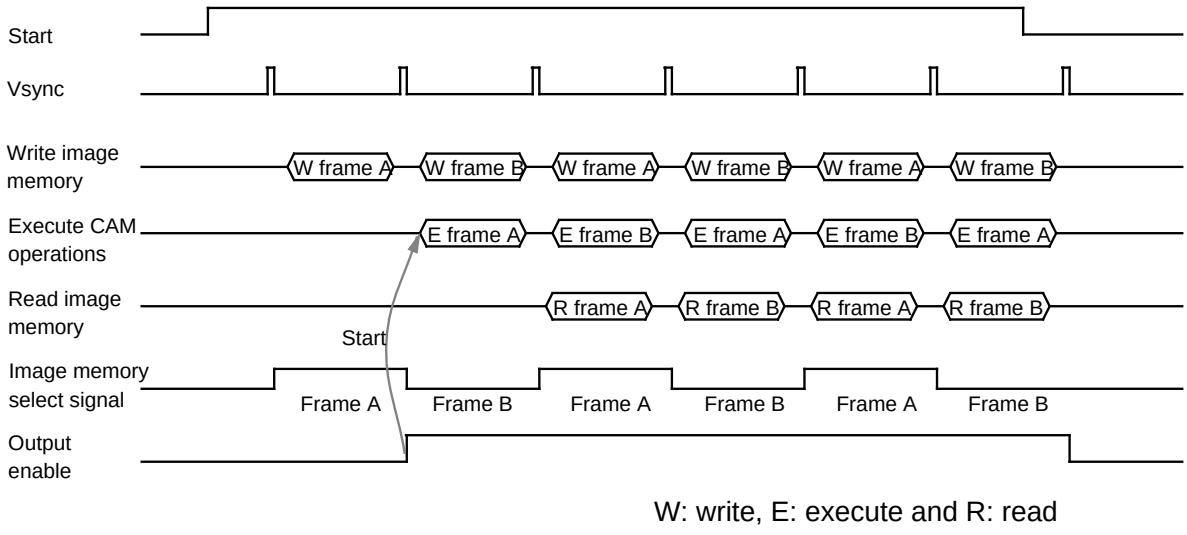
Figure 3.12 shows a control signal-flow graph and the board controller timing chart. When the system power is on, the board controller is set initially in the "idle" state. When the start signal is sent from the host PC through the PCI bus, the state moves to "ready". After receiving the first video synchronous signal (vsync), input image data from a video camera starts to be transferred to the frame A area of the on-board memory. When the second vsync is detected, input image data from the video camera are transferred to the frame B area. Simultaneously, data stored in frame A is sent to the CAM array, where various kinds of image processing are undertaken. When the third vsync is invoked, the output image is sent to a video monitor or PC and input image data from the video camera are transferred to the frame A area again. During this period, the CAM array processes the image data on frame B. This processing continues until the start signal is off. This control technique allows one port of the on-board memory to be shared without causing data destruction.

The board also has peripheral chips for PCI/IF and NTSC video encording/decoding. PLX PCI9060 is used to handle the PCI bus. PCI9060 [57] is a PCI Version 2.1 compliant bus master interface for adapter boards and embedded systems. The programmable local bus of the chip can be configured so that it directly connects a wide variety of processors, controllers and memory subsystems. It also provides two independent chaining DMA channels with bidirectional FIFOs supporting zero wait state burst transfers between a host and a local memory with a maximum data transfer rate of 132 MB/sec. By using the burst transfer function, input images, processed images and data can be transferred to or from a PC at video rates.

I used a Philips SAA7111 (video input processor) and an SAA7199 (digital video encoder) for video decoding and encoding, respectively. The CMOS circuit SAA7111, analog front-end and digital video decoder, is a highly integrated circuit for desktop video applications. This decoder is based on the principle of line-locked clock decoding and is able to decode the color of PAL and NTSC signals into CCIR-601 compatible color component values. The SAA7199 encodes digital base-band color/video data into analogy



a) Control signal flow graph of the board controller



b) Control flow timing chart

Figure 3.12: Control signal flow graph and timing chart of the board controller.

Y, C and CVBS signals. Pixel clock and data are line-locked to the horizontal scanning frequency of the video signal.

Since PCI bus and NTSC video interfaces are embedded in this board, a compact image-processing platform can be built simply by connecting the board to a PC and a video camera.

Figure 3.13 shows a photograph of the proof-of-concept prototype CAM² board. To minimize the noise effect, the analog circuits, video input processor and digital video encoder are embedded in a daughter board on the reverse side.

3.7 Conclusion

In this chapter, I described a 1-Mb CAM LSI for a CAM² CA cell. A scheme that combines one-dimensional (intra-block) and two-dimensional (inter-block) physical chip structures and comb data and clock distribution made it possible to develop a large capacity CAM that can handle 128 × 128 pixel images in parallel. Moreover, advanced functions, such as bi-directional hit-flag shift, hit-flag counting and search or parallel writing with an upward/downward mask shift, made it possible to carry out various types of two-dimensional pixel-parallel image processing including global data handling. I fabricated a chip that is capable of operating at 56 MHz and 2.5 V using 0.25-μm full-custom CMOS technology with five aluminum layers. A total of 15.5 million transistors were integrated into a 16.1 × 17.0 mm chip. The typical power dissipation is 0.25 W, and the chip exhibits a performance of 3 to 640 GOPS. This CAM LSI is expected to contribute significantly to the development of compact, high-performance image processing systems.

This chapter also described a prototype CAM² board that employs the CAM LSIs. The board consists of a highly-parallel PE array, FPGAs and some memory. The PE array is a two-dimensional array (2 × 2) of 1-Mb CAM LSIs, and can handle a 256 × 256 image. The FPGA generates various command sequences to perform practical image processing. In addition, PCI bus and NTSC video interfaces are also embedded in this board. Therefore, a compact image-processing platform can be built simply by

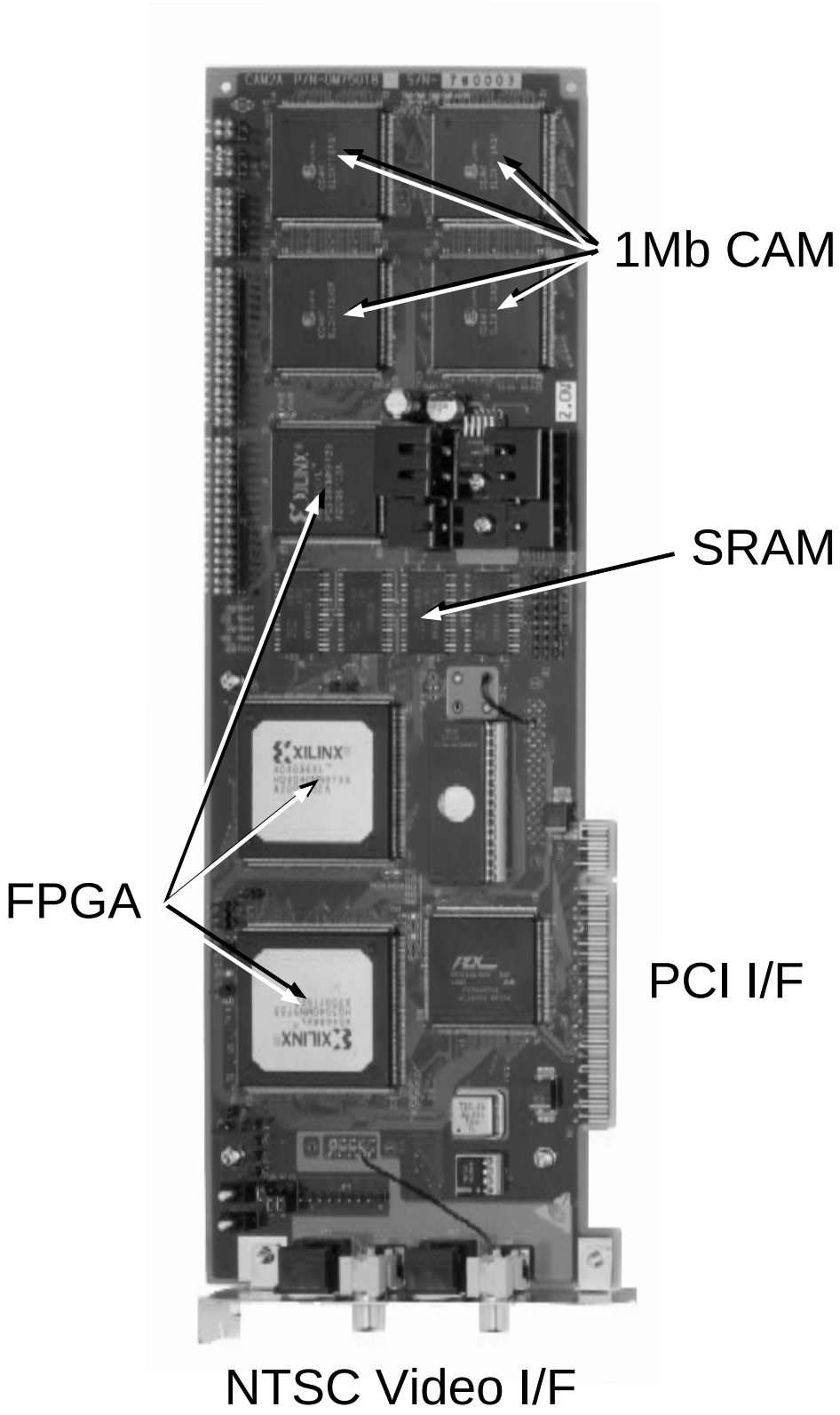


Figure 3.13: Prototype CAM^2 board with 256 $\times$ 256 CA cells.

connecting the board to a PC and a video camera. This prototype board demonstrates that an economically feasible image processing platform can actually be obtained.

Chapter 4

Application: Discrete-time CNN

4.1 Introduction

The discrete-time cellular neural network (DTCNN) [9], [10], which originated from cellular neural networks (CNN) [58] – [60], is a computer paradigm that fuses artificial neural networks with the concept of the cellular automaton (CA). Since a DTCNN's universality is equivalent to that of a CNN [61], it can handle many CNN image processing applications. For example, they include not only local image processing, such as edge detection; but also global image processing, such as hole filling [62], connected component detection [63] and skeletonization [64]. Furthermore, since a DTCNN is defined by difference equations instead of differential equations used in CNN definition, it is suitable for digital processing. Thus, the DTCNN is considered to be a promising computer paradigm for image processing. However, at present there are also no compact, practical computers that can process real-world images of several hundred thousand pixels at video rates. So, in spite of its great potential, DTCNNs are not being used for image processing outside the laboratory.

Conventional DTCNN architectures fall into two categories: an analog-array architecture [65] and a digital-pipeline architecture [66]. The analog-array type is the most natural one for DTCNN processing and provides very high level of performance. However, it is extremely difficult to develop a large capacity system that can process a practical image of several hundred thousand pixels because of problems with reliability, accuracy,

power, and production yield [67]. Therefore, only the analog-array CNN or DTCNN systems with very few cells [65], [68] have been successfully developed. On the other hand, the processing speed of the pipeline type is not high enough for many real-time applications. Furthermore, practical image processing requires various gray-scale image processing such as linear filtering [7] and gray-scale mathematical morphology [11]. But these architectures can not handle such processing. So, it is pretty clear that these conventional architectures are not good solutions for a compact, practical DTCNN processing platform.

In this chapter, I explain a DTCNN mapping and processing techniques based on CAM^2 in detail. For high-performance DTCNN processing, I devised the following four mapping techniques: 1) 1-bit coding of y, 2) table look-up multiplication, 3) addition of fewer bits and 4) convergence assessment using HFO. As for processing method, I discuss not only single-layer DTCNN but also multiple-layer DTCNN to perform more complex image processing. Furthermore, for rapid programming of DTCNN-based algorithms, a programming language and an application development environment for CAM^2 are presented. I also describe various performance evaluation results and image processing examples using the environment to demonstrate the usefulness of CAM^2 for DTCNN processing.

This chapter is organized as follows. Section 4.2 provides the definition of DTCNN. Then, the CAM^2-based DTCNN processing method is explained in Section 4.3. After describing the application development environment in Section 4.4, performance evaluation results and some examples of image processing combining DTCNN and other CA-based algorithms are presented in Section 4.5.

4.2 Definition of DTCNN

A DTCNN is defined by the recursive equations

$$x^c(k) = \sum_{d \in N_r(c)} a_d^c y^d(k) + \sum_{d \in N_r(c)} b_d^c u^d(k) + i^c \tag{4.1}$$

and

$$y^c(k) = f(x^c(k-1)) = \begin{cases} 1 & for\ x^c(k-1) \geq 0 \\ -1 & for\ x^c(k-1) < 0, \end{cases} \tag{4.2}$$

where x^c denotes the state of the cell c, y^c its output, and u^c its input. The state of a cell is calculated by convolutions of the inputs and outputs of neighboring cells within an r-neighborhood $N_r(c)$. The coefficients for the inputs and outputs are given by a_d^c and b_d^c, respectively. The output is obtained by thresholding the state. The value of i^c is constant and is used for adjusting the threshold. Just by changing the templates, a_d^c, b_d^c and i^c, various types of image processing can be performed.

Figure 4.1 shows the network structure of DTCNN. Applications for this network are especially in image processing, where, despite of the local connections, global tasks can be solved because of the feedback between the cells. The applications are separated into two classes:

- Local image processing is achieved by way of a discrete convolution or correlation with an input pattern. Here, local features of an image are extracted. Examples for this type are edge detection, corner extraction and averaging filter operations [59].

- Global image processing is performed by a wave-like propagation of the signals traveling from cell to cell such as hole filling [62], connected component detection [63] and shadow creation [69].

The originally defined single layer architecture has been generalized in order to perform more complex tasks, such as skeletonization [64] and motion detection [70]. For that purpose, a multiple layer structure has been introduced, where also nonlinear and delay-type templates [71] are applied.

A DTCNN can be considered to be a kind of two-dimensional CA whose transition rules are a combination of the convolutions and the thresholding defined in Eqs. (4.1)

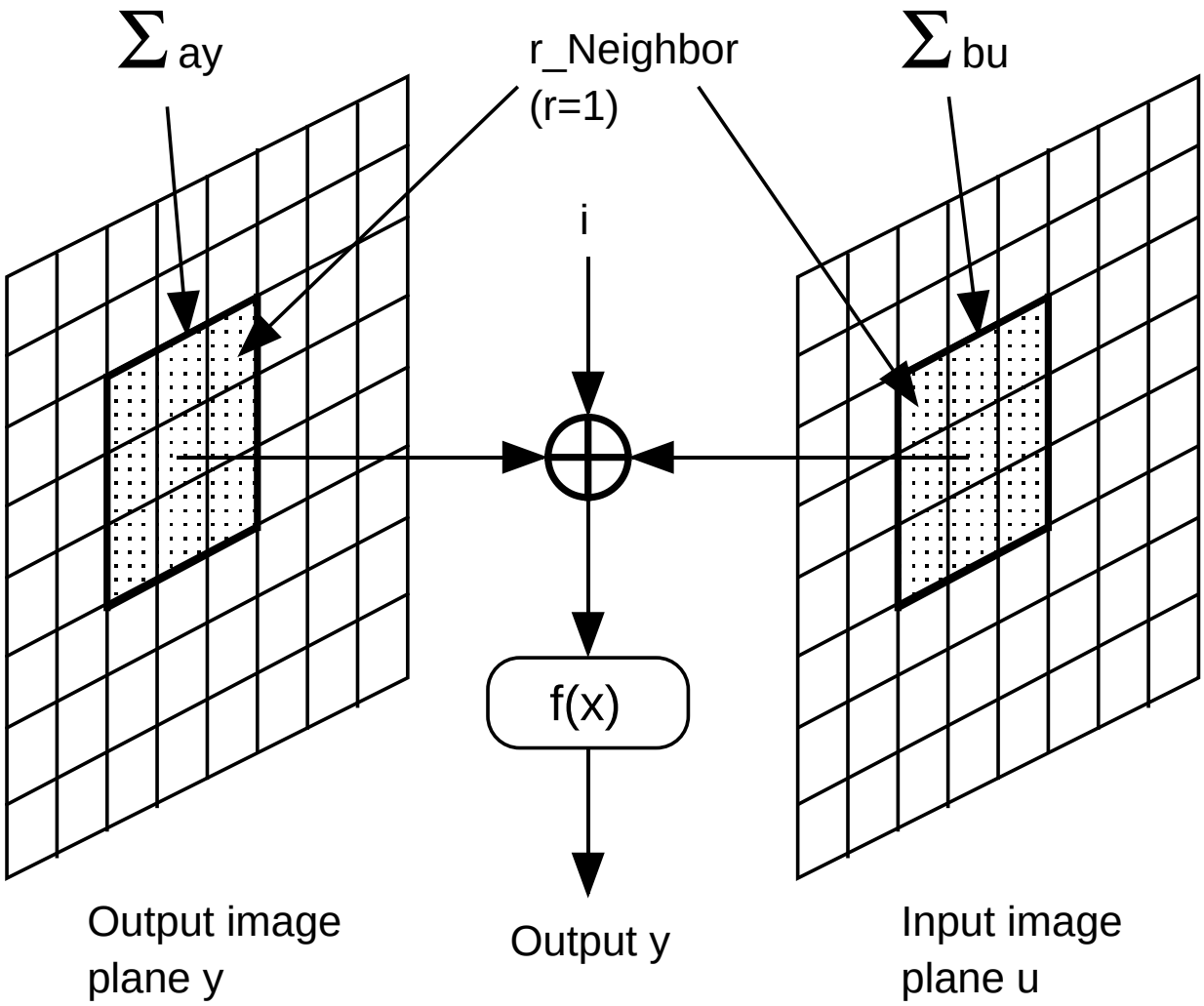


Figure 4.1: Network structure of DTCNN.

and (4.2). Therefore, it can be implemented with CAM^2. However, as they are, these equations contain very complex operations such as multiplication and take a long time to process. So, I devised effective mapping methods to shorten the time.

4.3 DTCNN Processing using CAM^2

4.3.1 Keys to DTCNN mapping

CA processing using CAM^2 is carried out by iterative operations of CA-value transfer and update as shown in Section 2.3. In order to carry out the above processing, the following three functions are absolutely essential:

- Maskable OR search (4.a)
- Partial & parallel write (4.b)
- Up & down shift of hit-flags (4.c)

For the search, the results are accumulated in hit-flag registers by means of OR logic. For the writes, the data are written into specific bit positions of multiple words for which the value of the hit-flag register is 1. For the shift, the hit flags are shifted to upper or lower words. Through the iteration of these operations, CA values are transferred and updated in a bit-serial, word-parallel manner.

The drawback of this scheme is that complex operations on many bits take a long time. Moreover, the time needed to transfer a CA value is proportional to the bit length of the value transferred. Thus, the keys to shortening the processing time are the transferring of fewer bits of a CA value and the simpler updating of fewer bits.

4.3.2 DTCNN mapping

Considering the above keys to CAM^2 processing, the following four mapping schemes are adopted.

1-bit coding of y

The first scheme is for CA-value transfer. Since CA-value transfer for y must be carried out for every transition, it can be a time-consuming process. To shorten the time, 1-bit coding of y is adopted. Since y only takes two values, 1 or -1 as shown in Eq. (4.2), this scheme is appropriate. In the scheme, "$y = 1$" and "$y = -1$" are coded to "1" and "0", respectively.

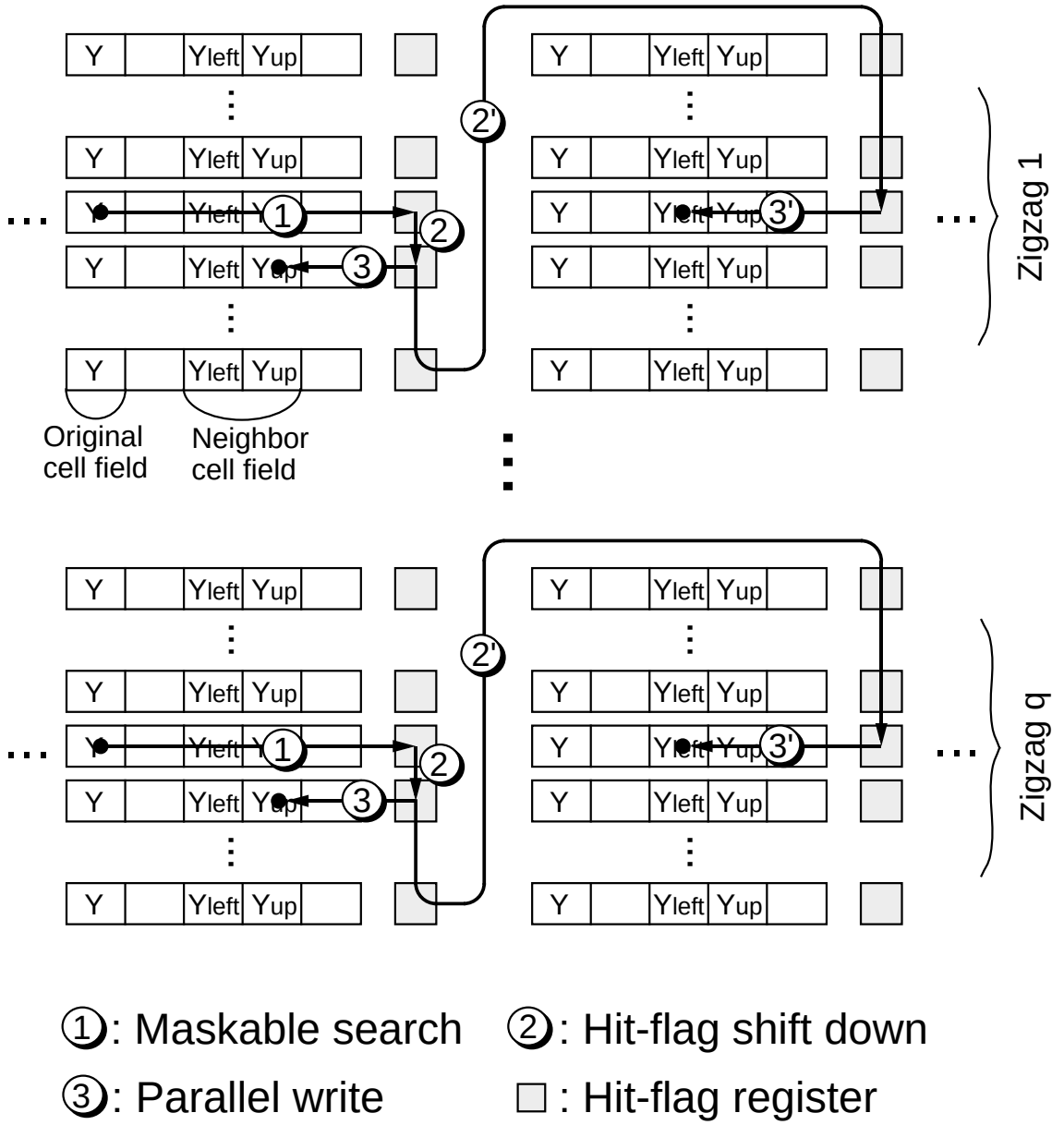


Figure 4.2: Example of CA-value (y) transfer.

Figure 4.2 shows an example of the CA-value transfer for DTCNN output. Here, y values of each cell are transferred to CA cells on the right and below and are stored in the neighbor cell field (Y_{left} and Y_{up}). As this example shows, the CA-value transfer can be carried out by using the iteration of maskable search (4.a), hit-flag shift (4.c) , and parallel write (4.b). Since the processing is carried out in a bit-serial manner, the number of CA-value transfer cycles for the 1-bit coding becomes half that for 2-bit coding in the 2's complement format, where "$y = 1$" and "$y = -1$" are coded to "01" and "11", respectively. Furthermore, the multiple-zigzag mapping cuts down the length of the hit-flag shift cycle that is the most time consuming. For example, the transfer cycle to a horizontally adjacent cell is $\frac{1}{q}$ compared with single-zigzag mapping.

Table look-up multiplication

To carry out Eq. (4.1), many multiplications of templates (a_d^c or b_d^c) and the outputs (y^d) or inputs (u^d) of neighbor cells are required. If a partial products addition method is adopted for the multiplication, the processing time becomes extremely long. For example, it takes more than a thousand cycles for 8-bit multiplication. Therefore, table look-up multiplication is adopted. In this scheme, multiplication is carried out by the iteration of maskable search (4.a) and parallel write (4.b). For example, $a_d^c \times y^d$ has only two possible results because a_d^c is a fixed value and y^d takes 1 or -1. The iteration is done only twice as shown in the following sequence:

1. Set search mask.
2. Set write mask.
3. Maskable search for cells whose y^d is 1.
4. Parallel writing of a_d^c to the hit cells.
5. Maskable search for cells whose y^d is -1.
6. Parallel writing of $-a_d^c$ to the hit cells.

Since CAM^2 can perform these sequences in only several cycles, this scheme takes much less time than one based on the addition of partial products.

Addition of fewer bits

Repeated additions are required for the convolutions in Eq. (4.1) and are also very time-consuming. The n-bit addition ($X + Y \Rightarrow X$) is executed in the following sequence:

1. Set search mask.
2. Set write mask.
3. Maskable search for cells whose $\{Xi, Yi, C\}$ is $\{0, 0, 1\}$.
4. Parallel writing of $\{1, 0\}$ to $\{Xi, C\}$ of the hit cells.
5. Maskable search for cells whose $\{Xi, Yi, C\}$ is $\{1, 0, 1\}$.
6. Parallel writing of $\{0, 1\}$ to $\{Xi, C\}$ of the hit cells.
7. Maskable search for cells whose $\{Xi, Yi, C\}$ is $\{1, 1, 0\}$.
8. Parallel writing of $\{0, 1\}$ to $\{Xi, C\}$ of the hit cells.
9. Maskable search for cells whose $\{Xi, Yi, C\}$ is $\{0, 1, 0\}$.
10. Parallel writing of $\{1, 0\}$ to $\{Xi, C\}$ of the hit cells.
11. Repeat 1-9 from LSB ($i = 1$) to MSB ($i = n$).

In the sequence, C field is used for storing carry. Figure 4.3 shows examples of the maskable search (4.a) in step 3 and the parallel write (4.b) in step 4.

As this sequence shows, the processing time for the additions is in proportion to their bit length (10 cycles per bit). Therefore, I limit the number of bits that must be added to match the dynamic range of x in order to process them faster. For example, since the dynamic range of x for processing hole filling [62], whose templates are shown in Fig. 4.4, is from -11 to 9, 5-bit addition is used to calculate the convolutions.

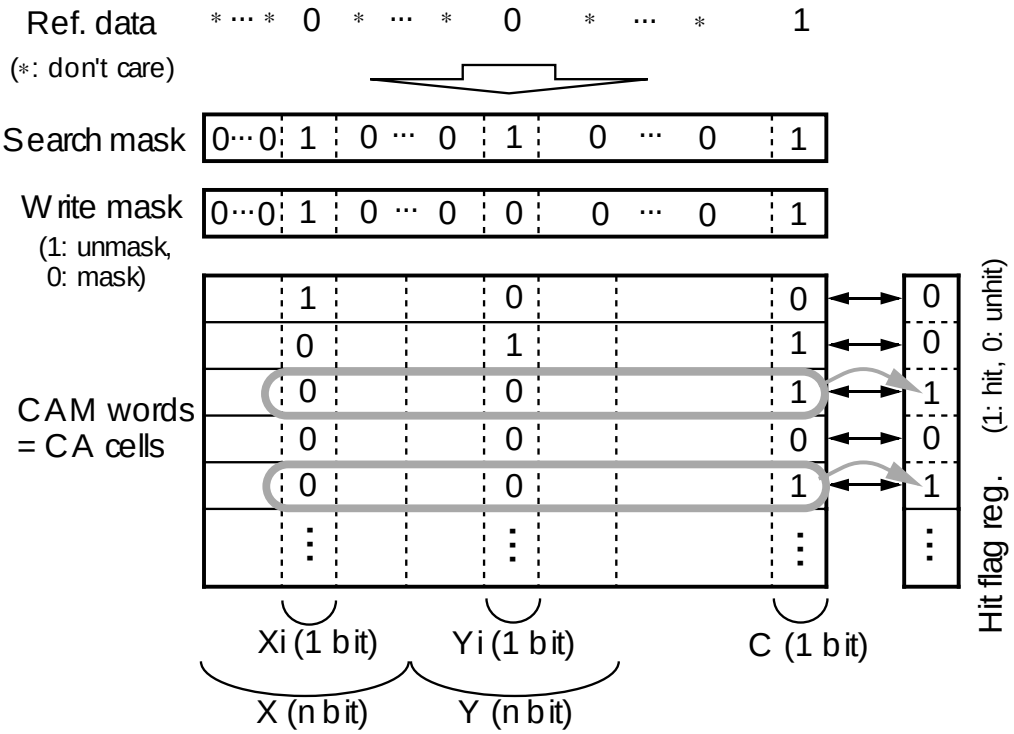


Figure 4.3: Example of maskable search for addition.

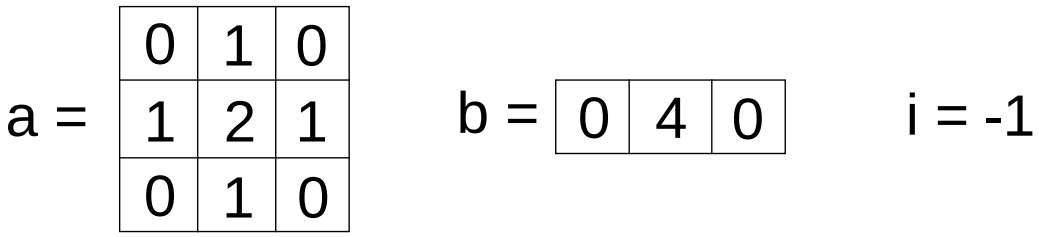


Figure 4.4: DTCNN template for hole filling.

Convergence assessment using HFO

The final mapping technique is for convergence assessment. The period of DTCNN dynamics changes according to the geometric features of the input image. So, convergence assessment is required to eliminate redundant transitions. In the convergence assessment, I must examine whether all cell values are equal to the correspondent or not. To execute this effectively, I output a hit flag (HFO) that is the logical OR of all the hit-flag registers.

The convergence assessment using the HFO is executed in the following sequence:

1. Set search mask.

2. Set write mask.

3. Maskable search for cells whose $y(k)$ is 1 and $y(k-1)$ is -1.

4. Maskable search for cells whose $y(k)$ is -1 and $y(k-1)$ is 1.

5. Convergence assessment (If "$HFO = 0$", then end DTCNN. Otherwise repeat new transition.).

In the sequence, first I use maskable search (4.a) to check each cell and see if $y(k)$ equals $y(k-1)$. Then, I assess the convergence by examining the value of HFO. When convergence occurs, the value is zero. CAM^2 can perform these sequences in only several cycles.

4.3.3 DTCNN processing procedure

Through the combination of the four mapping techniques, DTCNN can be implemented efficiently with CAM^2. Next, the DTCNN processing procedure employing these techniques is explained. As an example of the processing, I use hole filling with the template shown in Fig. 4.4.

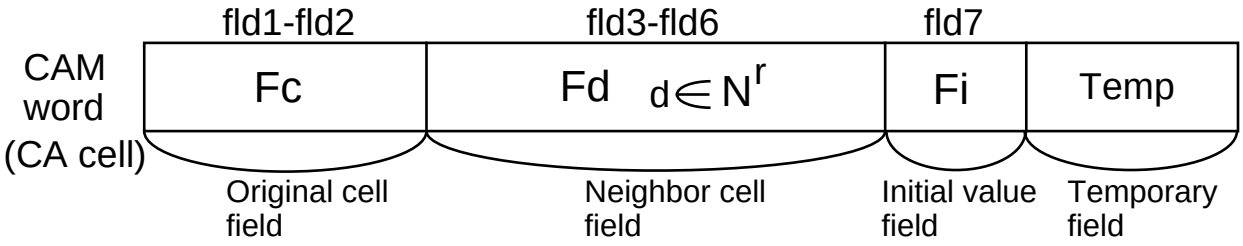


Figure 4.5: CAM word configuration for DTCNN processing.

The CA cell (= CAM word) configuration of CAM^2 for hole filling is shown in Fig. 4.5. Each CAM word has original cell fields F_c, neighbor cell fields F_d, an initial value field F_i, and a temporary field. The original cell fields store x^c and y^c of the original cell. The neighbor cell fields store $a_d^c \times y^d$ of left, right, upper and lower cells. The initial value field stores the initial value, which is fixed. The temporary field is used for storing carry, flags, and so on.

The hole filling is executed in the following sequence:

1. Load all pixel data of input image into F_c of correspondent CA cells of CAM^2.
2. Calculate $\sum b_d^c u^d(k) + i^c (= 4 \times u^d - 1\ )$ and store it to F_i.
3. Initialize y^c ($y^c(0) = 1$).
4. Transfer y of four neighbor cells (right, left, down and up cells) to correspondent F_d.
5. Calculate x^c according to Eq. (4.1) and store it to F_c.

6. Update y^c according to Eq. (4.2) and store it to F_c.

7. Repeat 4-6 until convergence occurs.

8. Read out y^c of all CA cells.

The data transfer in step 4 can be effectively performed using the intra-CAM and inter-CAM transfer described in Section 2.3. The convolutions in step 5, the thresholding in step 6, and the convergence assessment in step 7 can be effectively executed using the schemes mentioned in section 4.3.2.

4.3.4 Multiple-layer DTCNN processing

Not only a single-layer DTCNN but also a multiple-layer DTCNN with various templates is required to perform more complex image processing [9]. The multiple-layer DTCNN has three elementary modes for interconnecting multiple layers as shown in Fig. 4.6.

The cascade and feedback modes can be easily carried out by applying the templates (TEM1-TEMx, TEM1-TEMy) one by one using a sequence similar to that mentioned in section 4.3.3. On the other hand, since CAM^2 can perform only one DTCNN processing at a time, the parallel modes cannot be executed, as they are. So, I employ the following processing procedure.

The CA cell (= CAM word) configuration of CAM^2 for the parallel mode is shown in Fig. 4.7. Each CAM word has result fields F_r in addition to processing fields consisting of original cell fields F_c, neighbor cell fields F_d, an initial value field F_i, and a temporary field. The processing is executed in the following sequence:

1. Load all pixel data of input image into F_c of correspondent CA cells of CAM^2.

2. Calculate the DTCNN for a template (TEM1) using processing fields.

3. Store the result into a certain bit position of F_r.

4. Calculate the DTCNN for new templates (TEM2-TEMz) using processing fields.

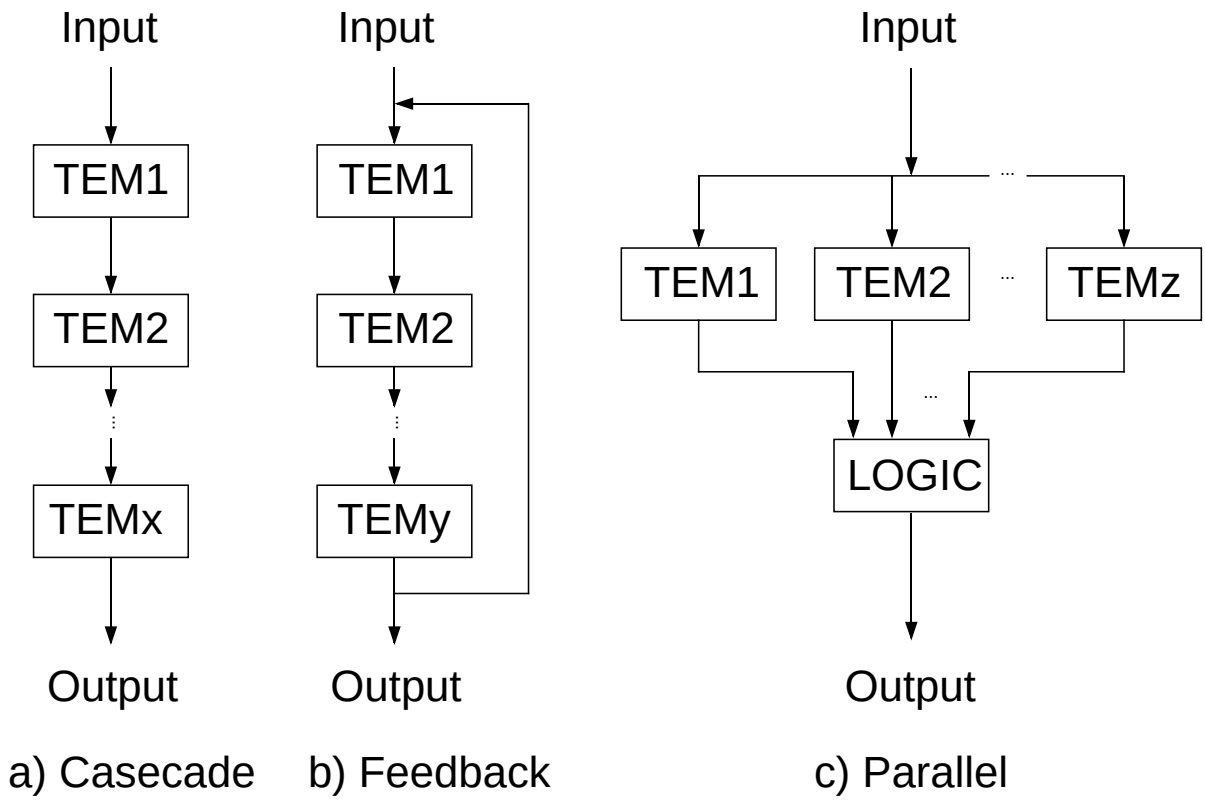


Figure 4.6: Various modes of multiple-layer DTCNN.

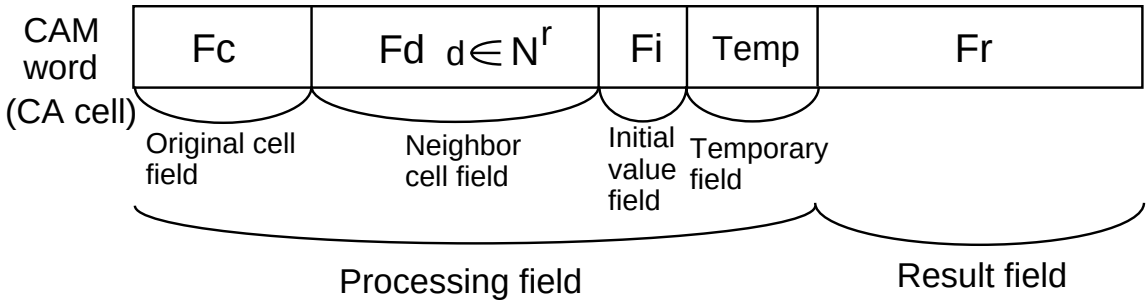


Figure 4.7: CAM word configuration for parallel mode.

5. Store the result into the other bit position of F_r.

6. Repeat 4-5 until all templates are calculated.

7. Perform a logic operation using all F_r data.

8. Read out the result for all CA cells.

In the prototype CAM^2 system described in Section 3.6, 64 bits are assigned to one word. Since from 30 to 40 bits are required to carry out various DTCNN processes, the number of bits which can be assigned to F_r is about 30. Only one bit is required to store one DTCNN result. Therefore, a parallel mode with about 30 templates can be executed.

4.4 Application development environment

To execute DTCNN processing using CAM^2, control logic for generating the various command sequences described in section 4.3.3 and 4.3.4 must be mapped into the FPGA on CAM^2. For efficient mapping, a programming language (CAM^2_PL) and an application development environment (CAM^2_ADE) for CAM^2 have been developed.

Figure 4.8 shows the feature of CAM^2_ADE. CAM^2_PL includes various arithmetic and logic operations, such as addition and logical OR; various associative operations, such as maskable search and parallel write; and some control-flow statements, such as repeat and while. It can easily handle a variety of CA algorithms.

CAM^2_ADE consists of a compiler and a simulator. The compiler compiles CAM^2_PL, synthesizes the logic, and generates a netlist for the FPGA. The simulator reports simulation results, like processing speed, for various input images and test data. They are used for debugging and evaluation. CAM^2_ADE should significantly speed up system development

An example of the CAM^2_PL for hole filling is shown in Fig. 4.9. "m_search" means maskable search and "p_write" means parallel write. "trans1_drul" means 1-bit 4-neighbor data transfer and "add_data5" means the addition of 5 bits. Through a combination of these operations, the hole filling can be described in only 34 statements.

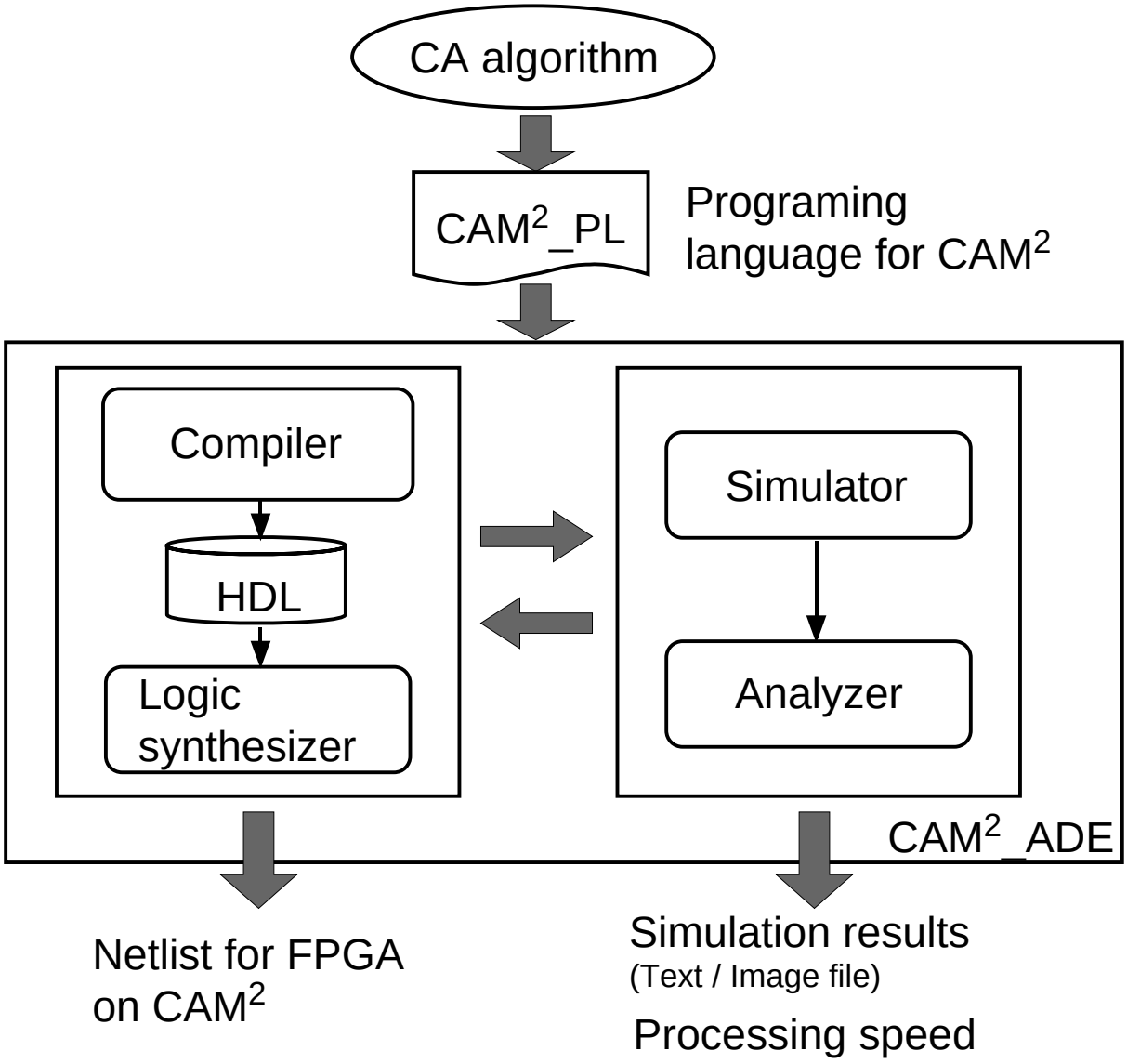


Figure 4.8: Feature of CAM^2_ADE.

```
;; DTCNN "hole-filling" for CAM^2
;;            written by CAM^2_PL

; Store binary image to fld1
(bin_data_in)

; Set outside cell data = 0
(set_0_load)

; Calc "SUM(bu)+i" and store to fld7
(m_search fld1 "01" "FF")
(p_write  fld7 "03" "FF")
(m_search fld1 "00" "FF")
(p_write  fld7 "1B" "FF")

; Set "y(0)" data to fld1
(field_id_set 1 "01")

; Repeat commands until HFO = 0
(while (HFO = 1) '(

   ; Copy data fld1 -> fld2
   (copy_8 fld1 fld2)
   ; 1-bit 4-neighbour data transfer
   (trans1_drul fld3 fld4 fld5 fld6)

   ; Calc original cell "ay(k)" data
   (m_search fld1 "01" "FF")
   (p_write  fld1 "02" "FF")
   (m_search fld1 "00" "FF")
   (p_write  fld1 "1E" "FF")

   ; Calc down cell "ay(k)" data
   (m_search fld3 "00" "FF")
   (p_write  fld3 "1F" "FF")

   ; Calc right cell "ay(k)" data
   (m_search fld4 "00" "FF")
   (p_write  fld4 "1F" "FF")
   ; Calc up cell "ay(k)" data
   (m_search fld5 "00" "FF")
   (p_write  fld5 "1F" "FF")
   ; Calc left cell "ay(k)" data
   (m_search fld6 "00" "FF")
   (p_write  fld6 "1F" "FF")

   ; Calc "x(k)" by summantion
   (add_data5 fld1 fld7)
   (add_data5 fld1 fld3)
   (add_data5 fld1 fld4)
   (add_data5 fld1 fld5)
   (add_data5 fld1 fld6)

   ; Update "y(k+1)" by thresholding
   (m_search fld1 "00" "10")
   (p_write  fld1 "01" "FF")
   (m_search fld1 "FF" "10")
   (p_write  fld1 "00" "FF")

   ; Convergence judgement
   (m_search fld1 fld2 "00" "01")
   (m_or_search fld1 fld2 "01" "00")
))

; Read binary image from fld1
(bin_data_out)
```

Figure 4.9: CAM^2_PL example for hole filling.

4.5 Evaluation

4.5.1 Processing performance

Table 4.1 shows the processing performance of one transition for various DTCNN templates. As described in Section 2.5, the functional hardware model of CAM^2 based on Verilog-HDL [55] was developed. This data comes from the Verilog functional simulator. In the evaluation, the system clock of CAM^2 was assumed to be 40 MHz. Image size was 512 $\times$ 512 pixels.

Table 4.1: Processing time for one transition.

Template	Processing time ($k \Rightarrow k+1$)
hole filling	14.1 μ sec
connected component detector	9.5 μ sec
concentric contours	14.1 μ sec
shadow creator	6.0 μ sec
decreasing object (8 neighbor)	18.3 μ sec

The table show that, on average, CAM^2 can perform one transition in about 12 μsec. This means that more than a thousand DTCNN updates can be carried out on the whole 512 $\times$ 512 pixel image at video rates (33 msec). So, I think CAM^2 has great potential for real-time image processing. On the other hand, if the same types of processing are executed on general sequential computers like PC or WS, about 1 second (on the Sparc station 20) is required for one transition. Thus, CAM^2 is extremely (a hundred thousands times) faster than such computers.

4.5.2 Image processing

Some examples of image processing using CAM^2 are shown in this section. These data were calculated by the CAM^2_ADE. Because of the simulation time limitation, a CAM^2 with 128 $\times$ 128 CA cells and a 128 $\times$ 128 pixel image was used. Since CAM^2 has scalability,

a 512×512 pixel image can be processed in the same time if a CAM^2 with 512×512 CA cells is used.

Single-layer DTCNN (hole filing)

Examples of the hole filling for two different input images, an 8 and an 8 in parentheses, are shown in Figs. 4.10 and 4.11, respectively.

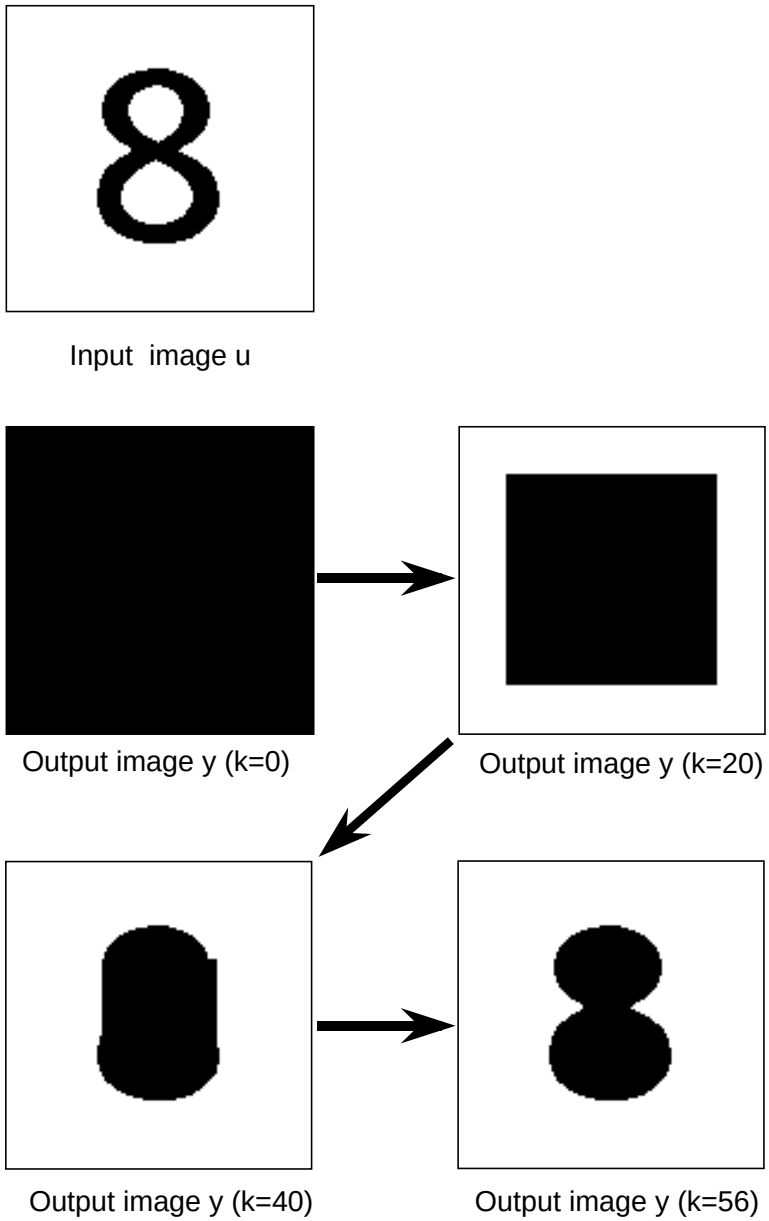


Figure 4.10: Hole filling (8).

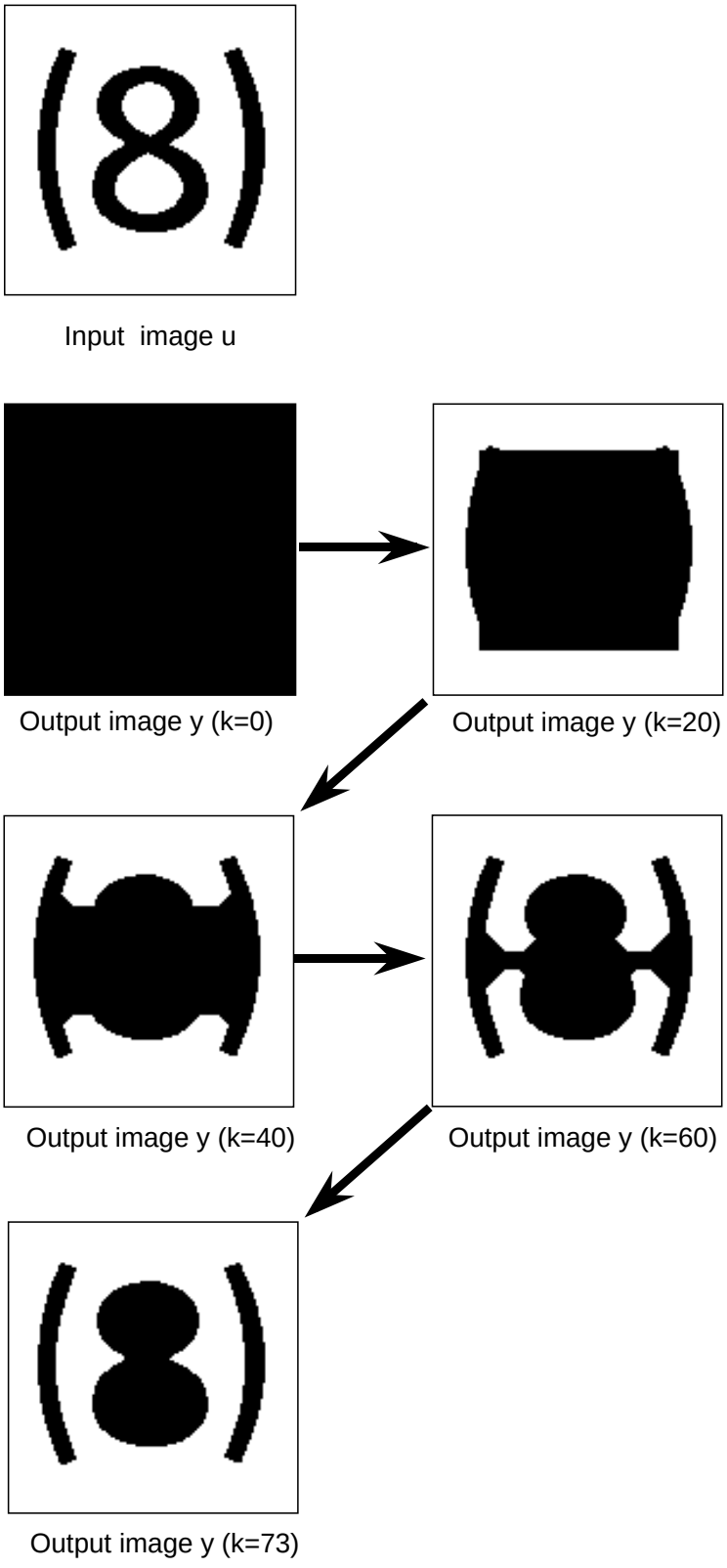


Figure 4.11: Hole filling (8 with parentheses).

By applying the template in Fig. 4.4 to the input images, initial output images ($k = 0$) are thinning gradually from the outside. And finally hole filled images are obtained.

The number of DTCNN updates necessary for completing the processing of the 8 is 56. On the other hand, 73 updates are required for the 8 in parentheses because the parentheses prevent quick convergence. Since CAM^2 can perform one transition for hole filling in 14.1 μsec (Table 4.1), the total processing times for the 8 and the 8 in parentheses are 790 and 1030 μsec, respectively.

As these examples show, the period of DTCNN dynamics changes according to the geometric features of the input image. Since conventional DTCNN machines do not have a mechanism to assess the convergence, they must continue the processing until the worst-case time. But CAM^2 easily performs the convergence assessment as explained in section 4.3.3. So, redundant transitions can be eliminated.

Combination of a multiple-layer DTCNN and other CA-based algorithms

Another image processing example is shown in Fig. 4.12. This is based on the combination of a multiple-layer DTCNN and other CA-based algorithms.

In the processing, sobel filtering and thesholding operations are used to perform edge detection, and the DTCNN is used for hole filling and decreasing objects. Center point detection is based on the skeletonization algorithm in [9]. The processing requires another CA-based algorithm for checking connecting pixels. This algorithm is given by the equations

$$x^c = \sum_{d=1}^{8}((-1)u_3^d u_3^{(d+1)mod8}) + i \tag{4.3}$$

$$y^c = f(x^c) = \begin{cases} 1 & for\ x^c \geq 0 \\ -1 & for\ x^c < 0 \end{cases} \tag{4.4}$$

where x^c denotes the state of the cell c, y^c its output, and u^c its input. Furthermore, the algorithm has a complex network combining the parallel and feedback modes. CAM^2 can also handle complex processing, such as center point detection.

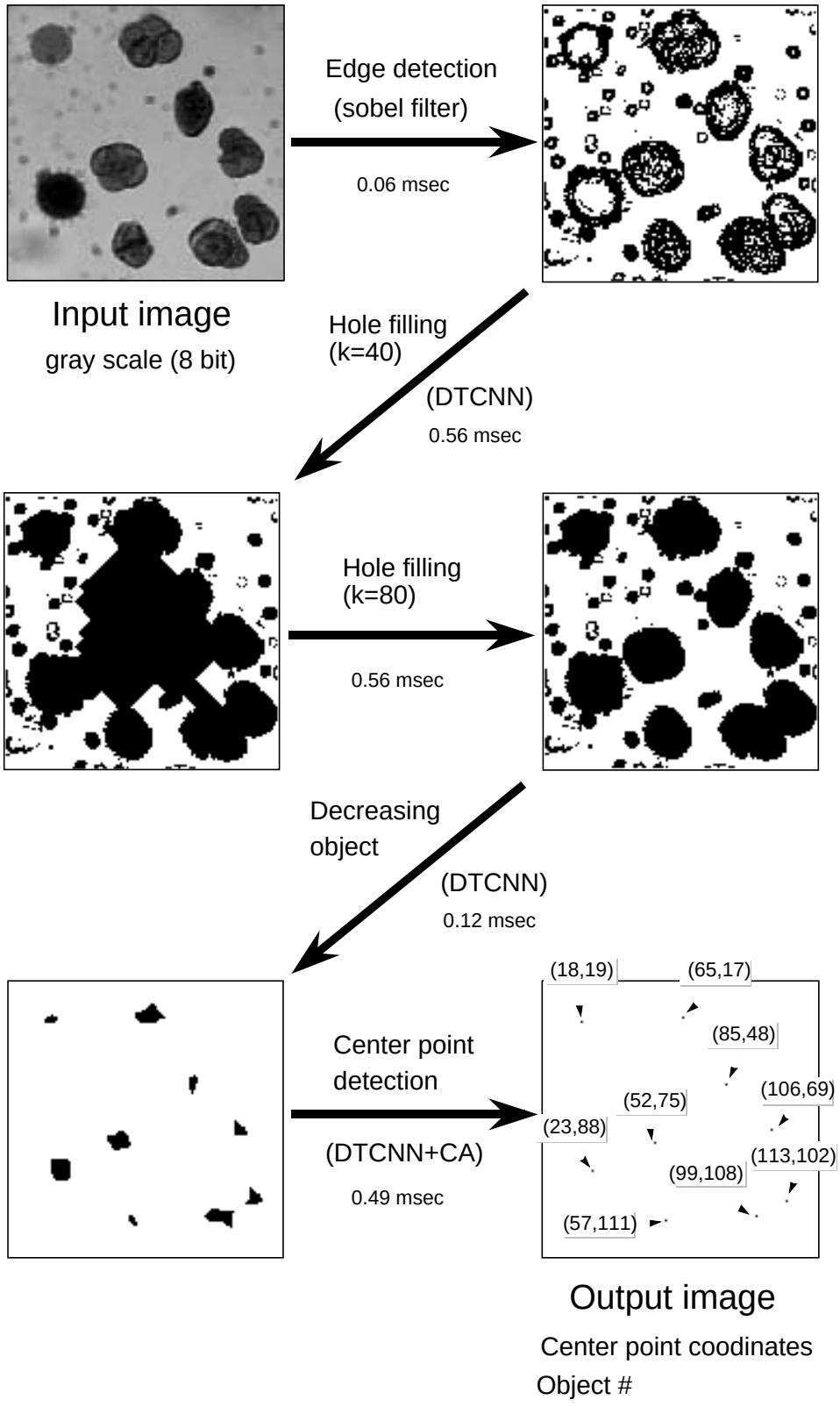


Figure 4.12: Image processing based on multiple-layer DTCNN.

By applying this processing to a gray scale image (8 bit, 128 × 128), the number of objects and the center point coordinates of the objects can be detected as shown in Fig. 4.12. It requires about 90 updates to all the CA cells. CAM^2 can do it in just 2 msec.

To complete the image processing, not only the DTCNN processing but also image data loading and retrieval processing are required. CAM^2 can also handle such processing effectively as show in Section 2.4. It takes about 0.1 msec for both the data loading and retrieval of a 128 × 128 image. And it takes 1.6 msec for a 512 × 512 image. Thus, the total processing time for the processing shown in Fig. 12 becomes less than 4 msec. This indicates that processing can be handled at video rate (33 msec).

As this example shows, CAM^2 has sufficient flexibility and performance. It can perform practical image processing that conventional machines cannot handle efficiently.

4.6 Conclusion

This chapter described a DTCNN processing method based on a highly-parallel two-dimensional cellular automata called CAM^2 and presented some evaluation results. For high-performance DTCNN processing, I devised the following four mapping techniques.

- 1-bit coding of y
- Table look-up multiplication
- Addition of fewer bits
- Convergence assessment using HFO

I also described multiple-layer DTCNN processing method to perform more complex image processing.

Evaluation results show that, on average, CAM^2 can perform one transition for various DTCNN templates in about 12 μsec. This means that more than a thousand DTCNN updates can be carried out on a whole 512 × 512 pixel image at video rates. CAM^2 performs practical image processing using not only a single-layer DTCNN, but also a

multiple-layer DTCNN with time-variant templates in combination with other CA-based algorithms.

CAM^2 enables us to realize a board-sized DTCNN universal machine that can process various real-world images of several hundred thousand pixels at video rates. Thus CAM^2 will widen the potentiality of DTCNNs and make a significant contribution to the development of various real-time image processing systems.

Chapter 5

Application: mathematical morphology

5.1 Introduction

Mathematical morphology has evolved in Europe since the 1960s as a set-theoretic method for image analysis, which uses concepts from algebra (set theory and complete lattices) and geometry (translation, distance and convexity). First, the development of mathematical morphology was motivated mainly by problems in quantitative microscopy. Then, the theoretical foundations of mathematical morphology, its main image operations (which stem from Minkowski set operations) and their properties, and a wide range of its applications were introduced systematically by Matheron [72] and Serra [11]. The image operations of mathematical morphology, or morphological filters, are more suitable for shape analysis than are linear filters. They are predominantly used for the following purposes:

- Image pre-processing (noise filtering, shape simplification).
- Enhancing object structures (skeletonizing, thinning, thickening, convex hull, object marking).
- Quantitative descriptions of objects (area, perimeter, projections, Euler Poincarè characteristic).

As a result of this pioneering work, mathematical morphology has become a powerful tool with various practical applications [73] – [77] in the field of biomedical image processing, metallography, geology, geography, remote sensing, astronomy, and automated industrial inspection.

There are three prerequisites for the fuller realization of the potential of mathematical morphology:

- Complex processing combining various morphological operations (including other operations, such as discrete-time cellular neural networks [9], linear filtering [7], and area calculation),
- Processing with large and complex structuring elements, and
- High-speed (real-time) processing.

The achievement of these goals requires hardware with extremely high levels of performance and high-frequency memory access. This also makes general-purpose sequential machines such as personal computers (PC) and workstations (WS) totally unsuitable.

To address these problems, a number of special-purpose morphology architectures have been proposed [78] – [82]. Most employ a pipeline technique in which a raster-scan image is fed sequentially into a processing element (PE) array and the morphological operations are carried out in parallel in each PE. Since the functions of the PEs and the network structure are fully tuned to morphology, other operations crucial to practical image processing cannot be performed. The fixed network structure also limits the size and shape of the structuring elements. Furthermore, there are at most several dozen PEs. This prevents the full use of the abundant parallelism (pixel order) of morphology, and, as a result, the processing speed of the pipeline type is insufficiently high for many real-time applications. Against this backdrop, it is pretty clear that none of these conventional architectures is suitable for building a morphology processing platform that satisfies the above three prerequisites.

In this chapter, I describe a morphology processing method that uses CAM^2. I propose new mapping methods designed to perform a wide variety of morphological operations for not only binary but also gray-scale images. For morphology processing with large and complex structuring elements, I devised a new transfer method, where the CA value can be transferred a long distance and to optional-position CA cells. I also present global operation procedures such as area calculation and null set assessment to obtain a pattern spectrum, which is a promising morphology-based algorithm.

This chapter is organized as follows. Section 5.2 provides the definition of mathematical morphology. This is followed, in Section 5.3, by a description of the morphology processing method. Section 5.4 discusses pattern spectrum processing and Section 5.5 presents performance evaluation results and examples of image processing combining morphology and other algorithms.

5.2 Definition of morphology

5.2.1 Dilation and erosion

Morphology falls into three categories [12]: set processing (SP), function and set processing (FSP), and function processing (FP). Each has two basic operations: dilation and erosion. Dilation ($\oplus$) and erosion ($\ominus$) are defined by

SP

$$A \oplus B^s = \{a + b : a \in A, b \in B^s\} \tag{5.1}$$

$$A \ominus B^s = \{a - b : a \in A, b \in B^s\} \tag{5.2}$$

FSP

$$(A \oplus B^s)(x) = max\{A(y) : y \in (B^s)_x\} \tag{5.3}$$

$$(A \ominus B^s)(x) = min\{A(y) : y \in (B^s)_x\} \tag{5.4}$$

FP

$$(A \oplus B^s)(x) = max\{A(y) + B(x - y) : y \in Z^2\} \tag{5.5}$$

$$(A \ominus B^s)(x) = min\{A(y) - B(x - y) : y \in Z^2\} \tag{5.6}$$

where A is the original image and B is the structuring element (SE).

As shown in the equations, dilation in SP employs the Minkowski addition of the original image and the structuring element. For erosion, addition is replaced by subtraction. FSP and FP are used for gray-scale image processing, which employs maximum and minimum operations.

5.2.2 Opening and closing

Erosion and dilation are not invertible transformations; if an image is eroded and then dilated the original image is not re-obtained. Instead, the result is a simplified and less detailed version of the original image.

Erosion followed by dilation creates an important morphological transformation called opening. The opening of an image A by the structuring element B is denoted by A_B and is defined as

$$A_B = (A \ominus B^s) \oplus B. \tag{5.7}$$

Dilation followed by erosion is called closing. The closing of an image A by the structuring element B is denoted by A^B and is defined as

$$A^B = (A \oplus B^s) \ominus B. \tag{5.8}$$

Opening and closing with an isotropic structuring element is used to eliminate specific image details smaller than the structuring element; the global shape of the objects is not distorted. Closing connects objects that are close to each other, fills up small holes, and smooths the object outline by filling up narrow gulfs. Meanings of “near”, “small” and

"narrow" are related to the size and the shape of the structuring element. Examples of opening and closing in SP are illustrated in Figure 5.1.

5.3 Morphology processing using CAM^2

5.3.1 Features of CAM^2 functions

CA processing using CAM^2 is carried out by iterative operations of CA-value transfer and update as shown in Section 2.3. In order to carry out them, CAM^2 has not only normal RAM operation, such as word reads and writes using addresses (5.a), but also the following three functions:

- Maskable OR search (5.b)
- Partial & parallel write (5.c)
- Shift up/down mode of hit-flags (5.d)

Although they are very simple, any type of CA operations can be carried out in a bit-serial, word-parallel manner through the iteration of these operations.

Since CAM^2 has only simple functions (thus allowing high-density implementation of CAM^2), the processing power of each CA cell (PE) is much lower than that of conventional highly-parallel machines, which support a variety of multibit arithmetic functions. Because of this drawback, processing time becomes longer as the complexity and bit length of operations increase. However, morphology requires only simple operations like logical OR and maximum, not complex ones like multiplication, which is commonly used in image processing filters. Furthermore, the dynamic range of morphological operations is fixed; for example, 1 bit and 8 bits for SP (set processing) and FSP (function and set processing), respectively. Therefore, the drawback mentioned above is not a serious obstacle to morphological processing. On the contrary, the simplicity is an advantage because it enables an enormous number of PEs to be built on a single CAM^2 chip, which allows the parallelism of morphology to be more fully exploited.

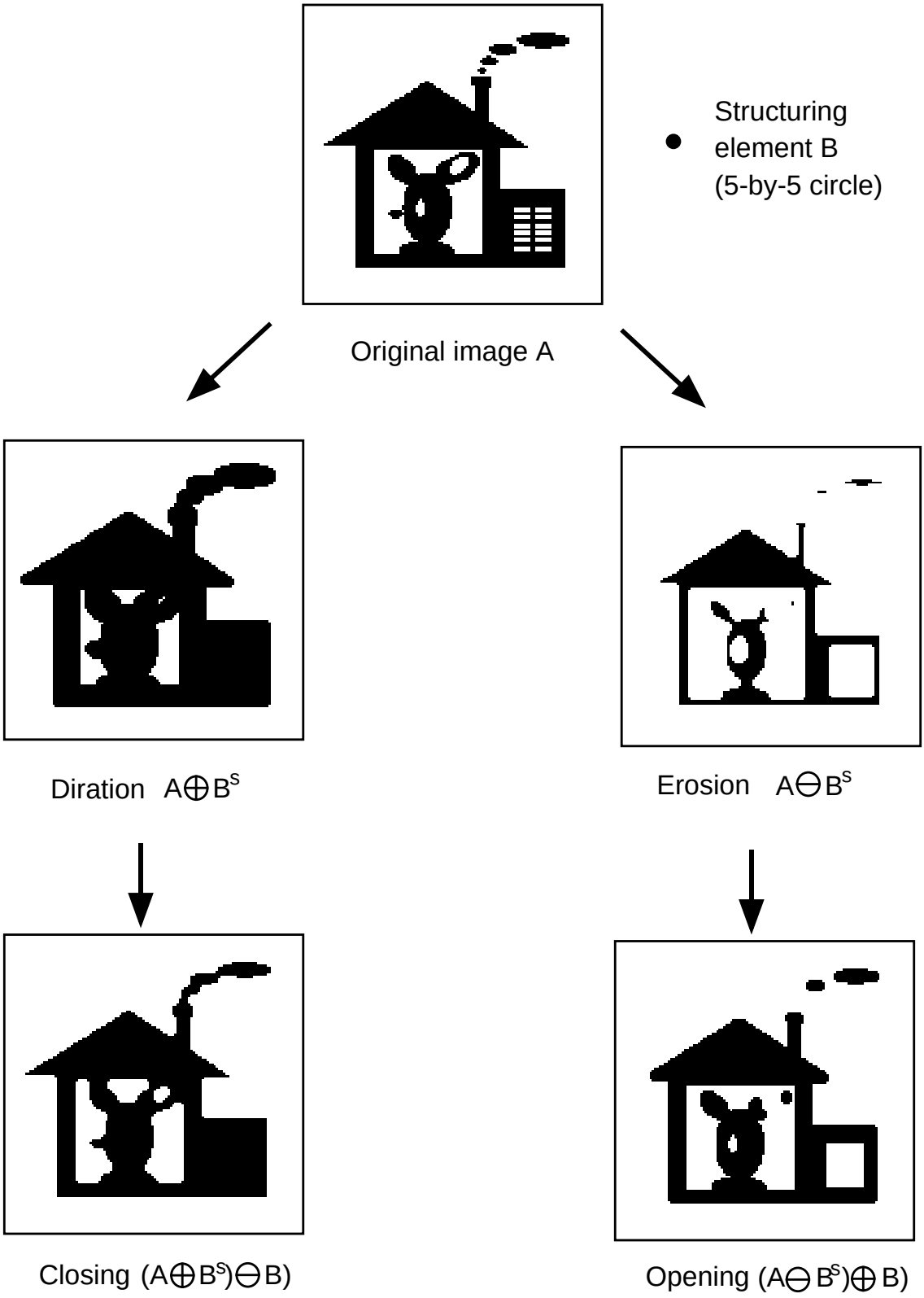


Figure 5.1: Examples of opening and closing (SP).

5.3.2 Morphology mapping to CAM^2

CAM^2 is a two-dimensional cellular automaton defined as follows:

- A set of two-dimensional cells (PEs) each with its own value.
- All the cells update their value simultaneously in discrete steps by a transition rule using the values of the original and the nearest neighbors.

For efficient execution of morphological equations by CAM^2, the following mapping scheme was devised:

- Map each pixel of the original image to a CA cell (PE) of CAM^2.
- The next value of the CA cells (the result of morphological operations) is determined by set operations (logical OR/AND, maximum, minimum, etc) for the values of the original and its neighboring cells. The cell location is defined by the structuring element.

If this mapping is adopted, morphology can be considered to be CA, in which neighbors are determined by the structuring element.

An example of dilation in FSP, in which the original image is gray scale and the structuring element is binary, is shown in Fig. 5.2. The set operation is maximum. For cell (7,7), the value in the pixel below is the maximum of the values of the original and neighboring pixels. So, this value is selected as the dilation result. And for cell (7,3), all the values are 0. So, the dilation result is also 0. Any type of morphological processing can be done in the same way.

5.3.3 Morphology processing method

In this section, morphology processing using CAM^2 is explained in detail. As an example, dilation and erosion in FSP with a rhombic structuring element are used.

Figure 5.3 shows the CAM word configuration for them. Each CAM word consists of an original image field, a dilation/erosion image field, neighboring cell fields, and a temporary

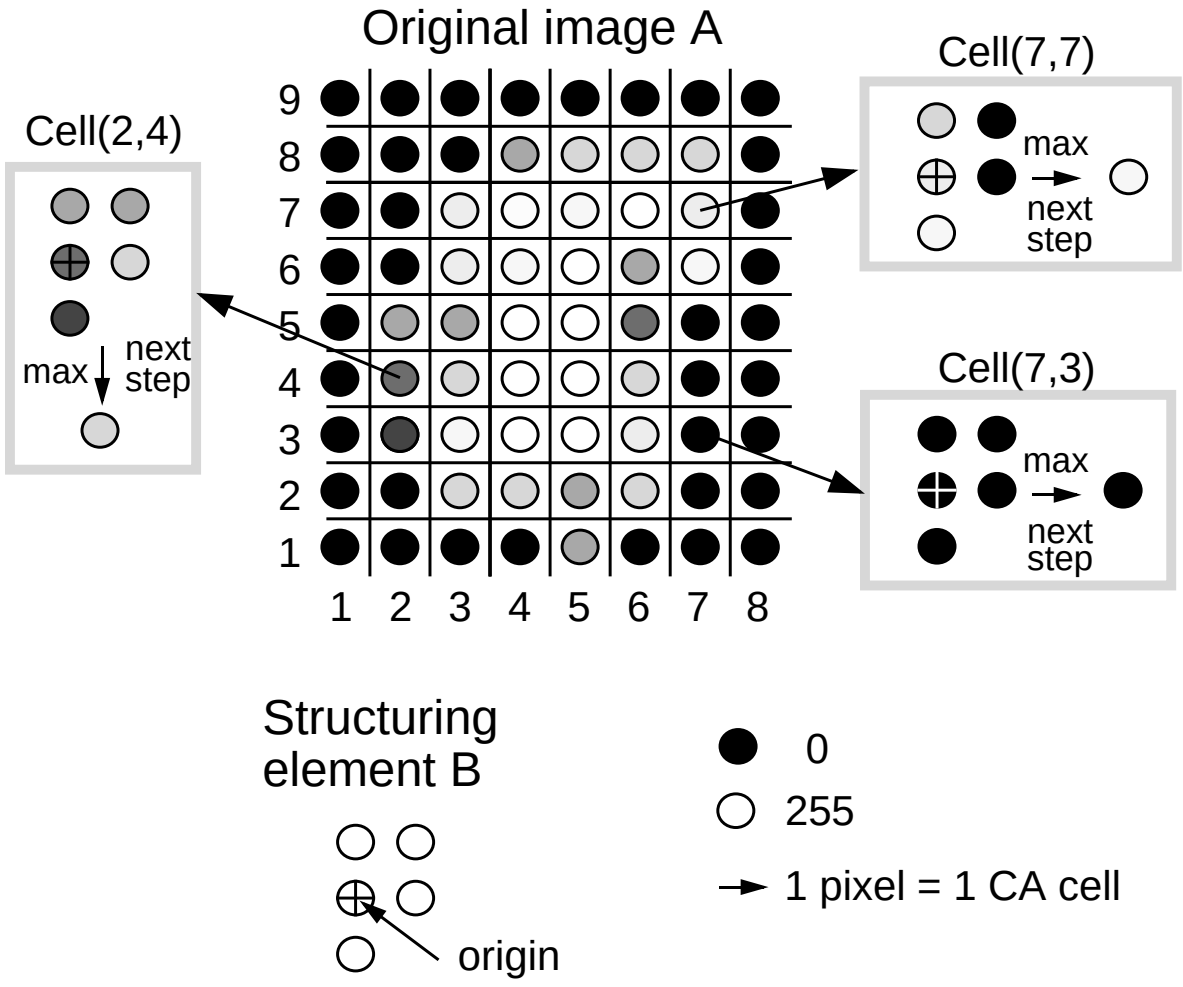


Figure 5.2: Example of dilation (FSP).

field. The original cell field (C) and the dilation/erosion image field ($C+$) store the value of the original image and dilation/erosion image, respectively. The neighbor cell fields (N_R, N_L, N_U and N_D) store values in the right (R), left (L), up (U), and down (D) cells. The temporary field is used for storing carry, flag, and so on.

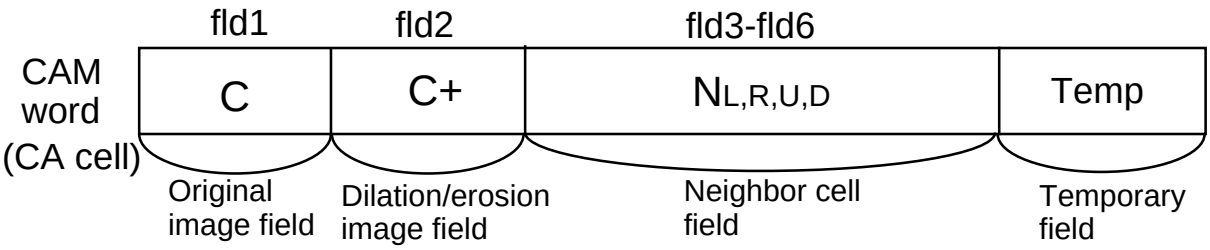


Figure 5.3: CAM word configuration for dilation/erosion (rhombus).

The dilation/erosion is executed in the following sequence:

1. Load all pixel data of the original gray-scale image into the C field of the corresponding CA cells of CAM^2.

2. Transfer the data of C to the $C+$ field of the same cell.

3. Transfer the data of C of four neighboring cells (right, left, upper and lower cells) to the corresponding neighboring cell fields.

4. Find out the maximum/minimum value among the data of $C+$ and N, and store it into the $C+$ field.

5. Read out the dilation result from the $C+$ field.

The image data loading and retrieval processing in steps 1 and 5 can be done by the normal RAM operations (5.a). Steps 2 and 3 are also effectively performed by the combination of intra-CAM and inter-CAM transfer. Step 4 is executed by the iteration of "greater than" or "less than" operations. Since "greater than" can be executed by the procedure shown in Section 3.3, the processing method of "less than" operation is explained in detail here.

Figure 5.4 shows the CAM word configuration for the "P-bit less than" operation, where the P bits of the X and Y fields are compared, and the one that's smaller is stored in the X field.

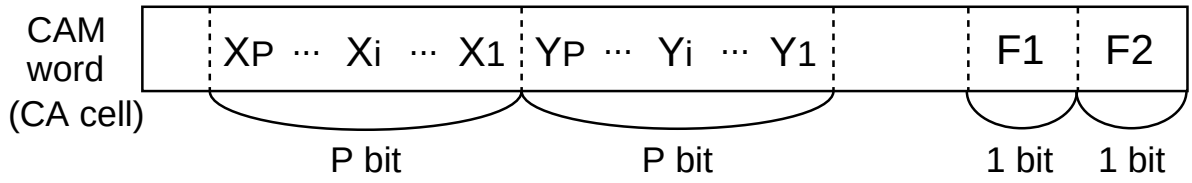


Figure 5.4: CAM word configuration for P bit less than operation.

The "less than" operation is executed in the following sequence:

1. Set search mask.
2. Set write mask.
3. Maskable search for cells whose $\{Xi, Yi, F1, F2\}$ are $\{1, 0, 1, 1\}$.
4. Maskable OR search for cells whose $\{Xi, Yi, F1, F2\}$ are $\{1, 0, 0, 0\}$.
5. Parallel writing of $\{0, 0, 0\}$ to $\{Xi, F1, F2\}$ of the hit cells.
6. Maskable search for cells whose $\{Xi, Yi, F1, F2\}$ are $\{0, 1, 1, 1\}$.
7. Parallel writing of $\{0, 0, 1\}$ to $\{Xi, F1, F2\}$ of the hit cells.
8. Maskable search for cells whose $\{Xi, Yi, F1, F2\}$ are $\{0, 1, 0, 0\}$.
9. Parallel writing of $\{1, 0, 0\}$ to $\{Xi, F1, F2\}$ of the hit cells.
10. Repeat 1-9 from MSB ($i = P$) to LSB ($i = 1$).

In the sequence, $F1$ and $F2$ are used for flags and are stored in the temporary field. The initial values of $F1$ and $F2$ are both 1. The condition $\{F1, F2\} = \{0, 0\}$ indicates that "$Y < X$" is determined and the condition $\{F1, F2\} = \{0, 1\}$ indicates the opposite.

As this example shows, the operation is carried out through the iteration of the maskable search and the parallel write. In the operation, the processing time is proportional to the bit length. It's 9 cycles per bit for the greater-than operation. However, since all the words are processed in parallel, the operations can be finished in an extremely short period of time.

5.3.4 Processing method for large and complex SEs

The size and shape of the structuring element are important factors in increasing the potential use of morphology. For the processing, the CA value must be transferred a long distance and to optional-position CA cells. To do this efficiently, the following method is devised.

The CAM word configuration is shown in Fig. 5.5. A CAM word consists of the original image field (C), processed image field ($C+$), and shift image fields (S_{UD1}, S_{UD2}, S_{RL}). The separated cell value is transferred efficiently and processed as follows:

1. Transfer the data of C of horizontal cells to the S_{RL} field using intra-CAM transfer.

2. Execute the set operation to $C+$ and S_{RL} if the correspondent structuring element is defined.

3. Repeat steps 1 and 2 until the horizontally defined structuring element runs out.

4. Transfer the data of C of vertical cells to S_{UD1} (after that S_{UD1} or S_{UD2} are used alternately) using inter-CAM transfer. This step is carried out at the same time as step 1.

5. Repeat steps 1 to 4 using S_{UD1} or S_{UD2} instead of C until the vertically defined structuring element runs out.

In the sequence, any shape of structuring element can be coped with by determining whether the set operation is executed or not according to the structuring element.

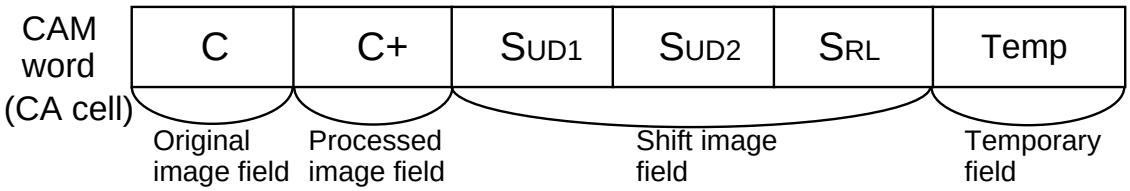


Figure 5.5: CAM word configuration for large and complex SEs.

5.4 Pattern spectrum processing

The pattern spectrum processing [83] has been proposed as a morphology-based algorithm. It is very useful for getting information on the global features of target objects, and some applications, such as a gender recognition [84], have been devised.

Figure 5.6 shows an example of pattern spectrum processing, where n and B show scales and structuring elements, respectively. $A(X)$ shows the area of image X. As shown in Fig. 5.6, to obtain a pattern spectrum, operations other than morphological ones are required. They are summarized as follows:

- Pixel-by-pixel subtraction $(X_{nB} - X_{(n+1)B})$.
- Area calculation (number of black pixels in $(X_{nB} - X_{(n+1)B})$ image).
- Null set (ϕ) assessment of $X_{(n+1)B}$.

The following mapping schemes provide an efficient way to do these.

5.4.1 Pixel-by-pixel subtraction

The first scheme is for pixel-by-pixel subtraction. Optional-bit-width subtraction can be performed at the rate of about ten cycles per bit by combining the maskable search (5.b) and parallel write (5.c), just as in the "greater than" operations described in Section 5.3.3. However, making use of the relationship of opening, $X_{nB} \geq X_{(n+1)B}$, shortens the processing time still more. This can be done in a sequence that takes only two cycles:

1. Set search mask.

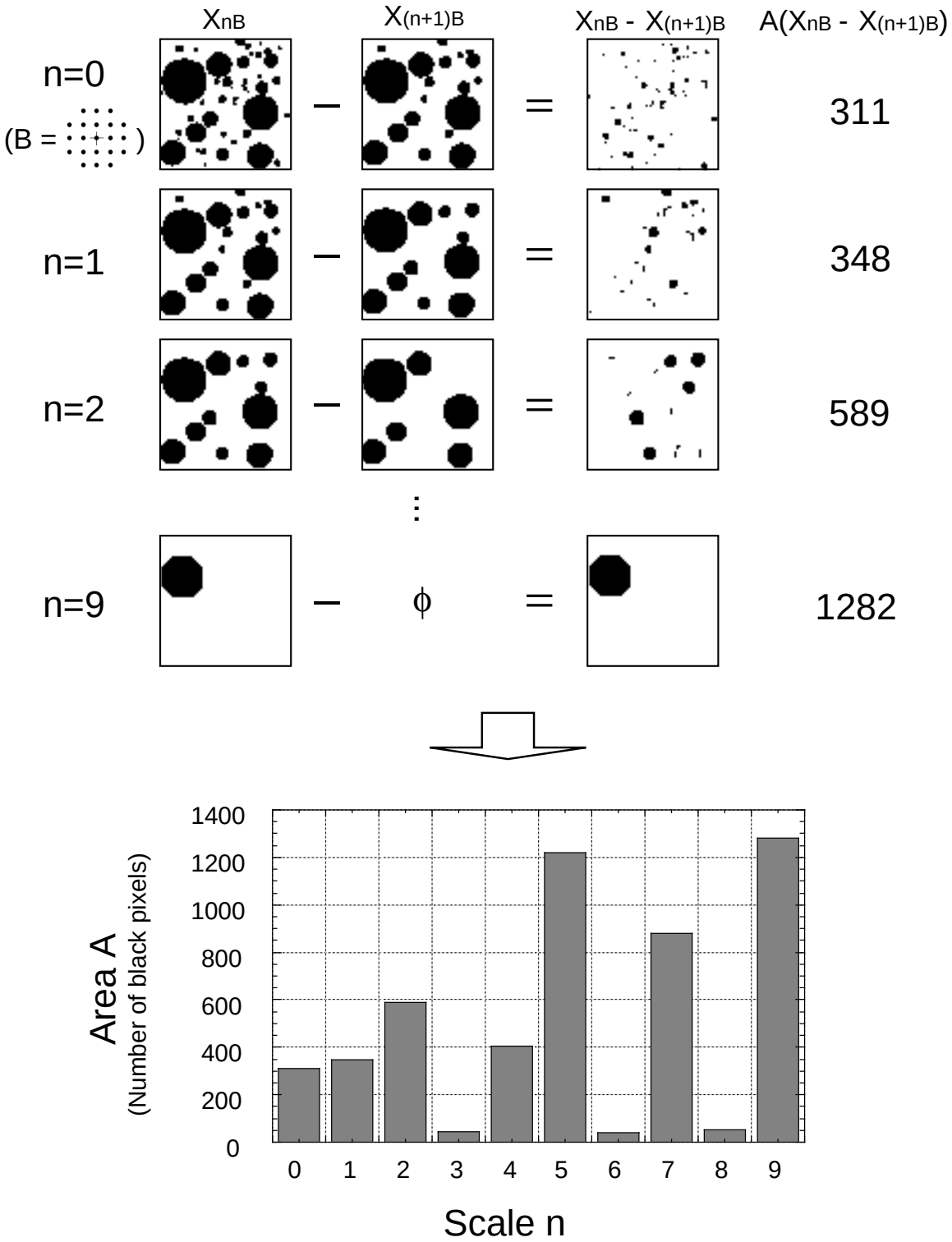


Figure 5.6: Example of pattern spectrum processing.

2. Maskable search for cells whose X_{nB} is 1 and $X_{(n+1)B}$ is 0.

A positive result for a particular cell is stored in its hit-flag register. Thus, the processing time can be shortened significantly by exploiting the features of target algorithms, even though the performance of each cell (PE) of CAM^2 is not very high, as mentioned before.

5.4.2 Area calculation

To calculate an area, the pixel values stored in all the cells must be summed up. Generally speaking, however, highly-parallel machines based on local operations, including cellular automata, cannot handle such global operation efficiently. Indeed, normal CAM has only one global network, which is the data I/O. So, to perform the calculation, data in each cell (word) must be retrieved through the I/O one by one and summed up using an external processor or some special circuits.

To address these problem, CAM^2 has both counters in each CAM block (to count the number of hit flags) and horizontal and vertical inter-CAM connection networks (for data transfer between adjacent CAM blocks), as shown in Fig. 5.7. These functions can be implemented just by changing the peripheral circuit of CAM, i. e., changing the memory cell part is unnecessary. Moreover, the counter can be shared with the pipeline register for the transfer. So, they can be implemented without degrading the high density of CAM.

The area calculation is done as follows:

1. Shift the hit-flag registers of CA cells in which the pixel values are stored by means of pixel-by-pixel subtraction and count the number of hit flags using the counters. (This yields the area of each CAM block.)

2. Sum up the areas of all the CAM blocks through the iteration of the inter-block transfer using the connection networks and addition using the maskable searches (5.b) and the parallel writes (5.c). (This operation moves through the area data in a tree-like fashion and stores the area of the whole image in a particular cell.)

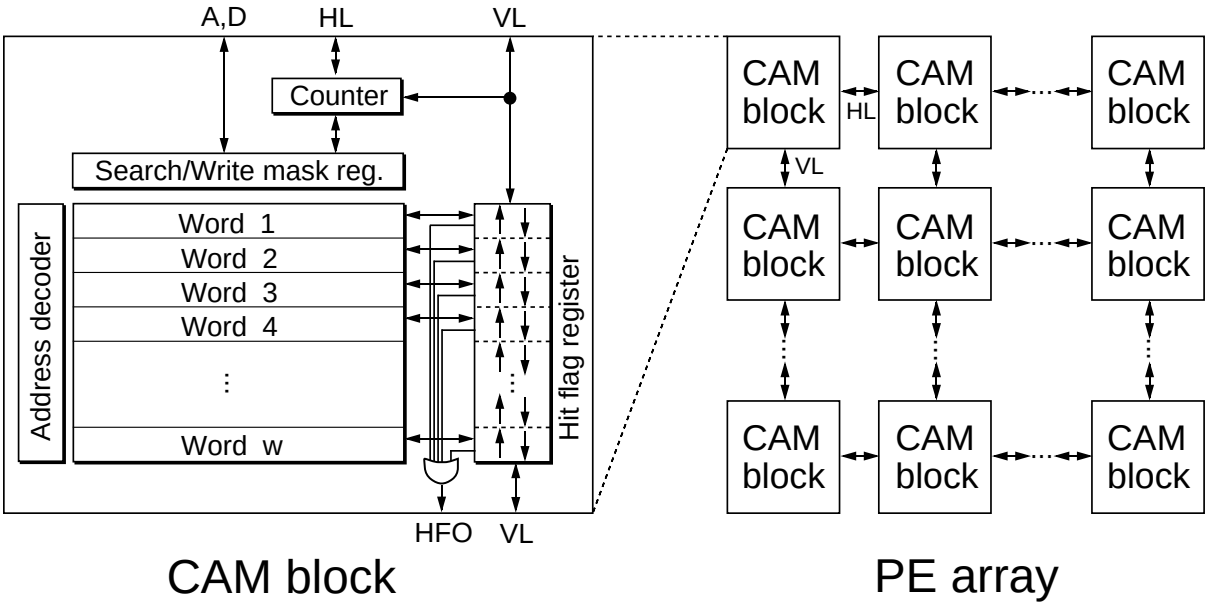


Figure 5.7: CAM structure for area calculation.

Step 1 of this sequence is carried out in parallel in each block. Moreover, the number of iterations in step 2 varies logarithmically with the number of blocks. So, an area calculation only takes a short time.

5.4.3 Null set assessment

The final mapping technique is for null set assessment. The scale when $X_{(n+1)B}$ becomes a null set changes according to the geometric features of the input image. So, null set assessment is required in order to eliminate redundant transitions. In the assessment, I must examine whether $X_{(n+1)B}$ values stored in all the cells become "0" or not. To do this efficiently, I output a hit flag (HFO) that is the logical OR of all the hit-flag registers as shown in Fig. 5.7.

The assessment using the HFO is executed as follows:

1. Set search mask.

2. Maskable search for cells whose $X_{(n+1)B}$ is 1.

3. Null set assessment (If "$HFO = 0$", then end the processing. Otherwise repeat the processing for a new scale.).

When all $X_{(n+1)B}$ values become 0, the result of the maskable search for all the cells becomes unhit. So, HFO becomes 0. A null set is assessed by examining the value of HFO. Since CAM^2 performs these sequences in only several cycles, the null set assessment is also finished in an extremely short period of time.

5.5 Evaluation

5.5.1 Processing performance

Morphology processing performance is evaluated in this section. I have already finished the design of CAM^2 and have described it in Verilog HDL [55]. This data comes from the Verilog functional simulator. In the evaluation, the system clock of CAM^2 was assumed to be 40 MHz. Image size was 512 × 512 pixels.

Table 5.1 shows the processing performance for one dilation of basic SEs shown in Fig. 5.8. Erosion is performed in almost the same time. Through a combination of these structuring elements, morphological operations with various-sized regular-shaped structuring elements, such as 3 × 3 square and 5 × 5 circle, can be performed. As shown in the table, CAM^2 performs one dilation within 30 μsec. This means that more than a thousand dilations can be carried out on the whole 512 × 512 pixel image at video rates (33 msec).

Table 5.1: Processing time for basic SEs (μsec).

	lin0	lin45	lin90	boxne	rhombus
SP	7.0	8.3	1.3	4.4	7.4
FSP	12.3	20.5	11.8	15.0	18.0
FP	18.7	26.9	18.2	23.5	28.6

Figure 5.9 shows the processing performance for one dilation of large SEs. In the

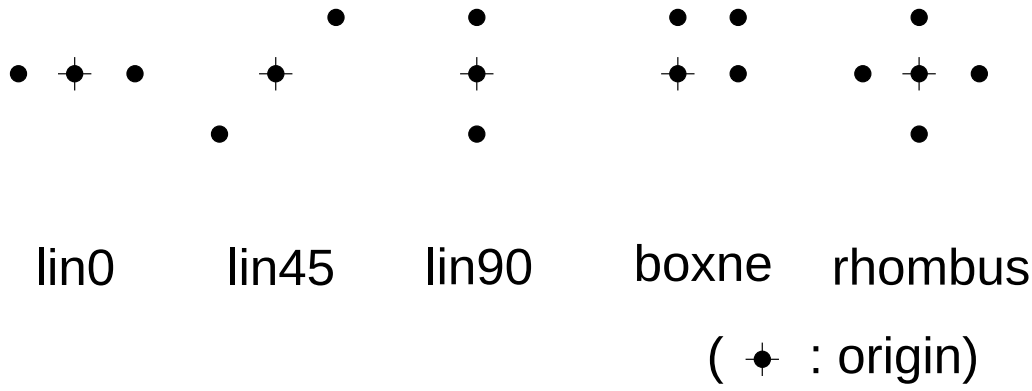


Figure 5.8: Basic structuring elements.

evaluation, regular square structuring elements with size L^2 were used. The figure shows that the execution time for one dilation is almost proportional to the size of the structuring element. Therefore, an extremely large structuring element with a size of about $100 \times 100 = 10^4$ can be handled at video rates.

In view of these simulation results, I think CAM^2 satisfies the three prerequisites mentioned in Section 5.1, and, therefore, has great potential for morphology processing.

5.5.2 Image processing

Some examples of image processing based on morphology are shown in this section. These data were calculated by CAM^2_ADE based on the Verilog functional simulator. Although these examples make it necessary to perform iterative morphological operations, the Verilog simulator takes an extremely long time to run. So, a CAM^2 with 128×128 CA cells and a 128×128 image were used here. Since CAM^2 has scalability, a 512×512 image can be processed in almost the same time (except for data loading and retrieval processing) if a CAM^2 with 512×512 CA cells is used.

As described in Section 4.4, CAM^2_PL includes various arithmetic and logical operations, such as addition and logical OR, and various associative operations, such as maskable search and parallel write. To describe various morphological operations easily, the following dedicated operations are added:

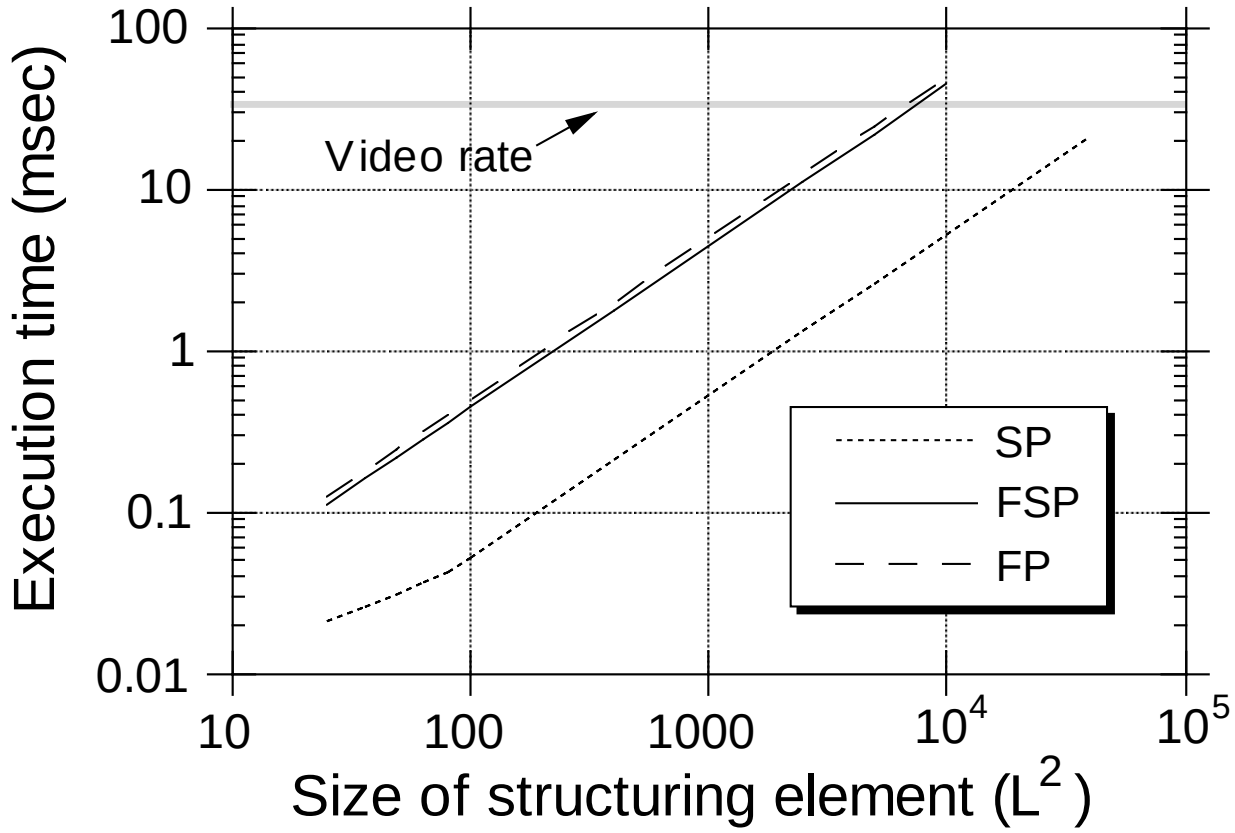


Figure 5.9: Processing time for large SEs.

- (dilation *type se*)
- (erosion *type se*)

where the morphology category (SP, FSP, and FP) and shape of the structuring element (rhombus, square, etc) are given in *"type"* and *"se"*, respectively. An example of CAM^2_PL is presented in Section 5.5.2.

Data loading and retrieval processing

The completion of image processing requires not only morphological processing but also data loading and retrieval processing. For loading, all the pixel data of the input image are loaded into the corresponding CA cells of CAM^2. For retrieval, the processed data are retrieved from the result fields of all CA cells.

Using parallel loading and partial word retrieval techniques shown in Section 2.4, CAM^2 can also handle such processing effectively. It takes about 0.1 msec for both the data loading and retrieval of a 128×128 image. The processing time needed for data loading and retrieval processing lengthens with image size. However, since it only takes 1.6 msec even for a relatively large image (512×512), more than 30 msec is available in which to perform morphology or other CA algorithms for real-time, or video rate (33 msec), applications.

Pattern spectrum

Figure 5.10 shows examples of pattern spectrum processing (SE: 5×5 circle) for two different images, in which one object has a crack and the other one does not. As shown in Fig. 5.10, since the size and shape of the objects in image 1 are uniform, the spectrum concentrates on scales 5 and 6. In contrast, since the object in image 2 has cracks, the spectrum is scattered. Since the features of the spectra are quite different, it is easy to distinguish them.

Table 5.2 shows the processing time for the images. As shown in the table, the processing time for opening increases with the scale because the size of the structuring

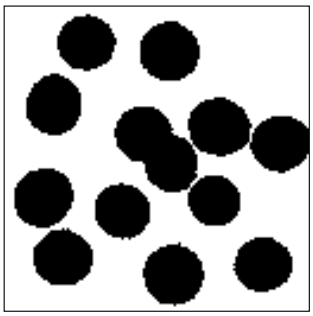
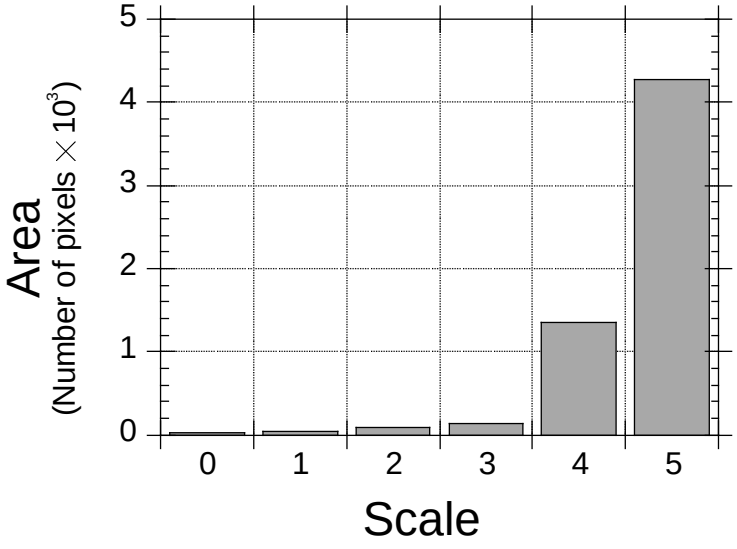


Image 1



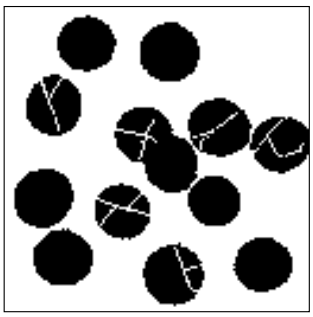


Image 2

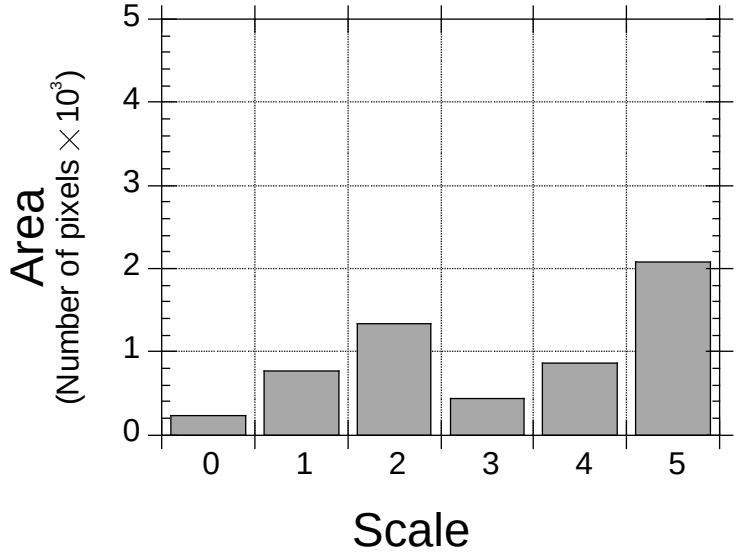


Figure 5.10: Pattern spectrum for images with and without crack.

element becomes larger as the scale increases. In contrast, the processing time for the area calculation is constant, and is about $\frac{1}{10}$ that of the conventional method using the data I/O and the external processor. It takes only 0.6 μsec per scale for the rest of the processing, such as pixel-by-pixel subtaction.

It takes about 1 msec for the whole pattern spectrum processing. When data loading time is included, the total time becomes 1.1 msec and 1.8 msec for 128 $\times$ 128 and 512 $\times$ 512 images, respectively. Thus, a pattern spectrum can also be obtained at video rates.

Table 5.2: Pattern spectrum processing time.

Processing	1 scale (μsec)	all scales (μsec)
opening	34.2 (n=0) - 204.2 (n=5)	714.2
area calc.	33.8	202.8
rest	0.6	3.6

Skeletonization

Skeletonization is an important technique for analyzing images, as it can be used to provide their medial axes. The morphological skeleton [12] $SK(A)$ of image A (with respect to structuring element B) is the finite union of disjointed skeleton subsets $S_n(A)$ and is defined by

$$SK(A) = \bigcup_{0 \leq n \leq N} S_n(A), \tag{5.9}$$

where

$$S_n(A) = (A \ominus nB^s) - (A \ominus nB^s)_B \tag{5.10}$$

and $n = 0, 1, 2, ..., N = max\{k : A \ominus kB^s \neq \phi\}$.

Figure 5.11 shows an example of morphological skeletons. I used four different 3-by-3 structuring elements (lin0, lin90, rhombus and square). As shown in the figure, the

shape of the skeletons varies according to the structuring elements. It shows the ability of morphological filters to extract different structuring information by using different structuring elements. The number of iterations N is from 11 to 17 and the procedure takes from 0.04 to 0.27 msec. Thus, CAM^2 can also obtain various morphological skeletons at video rates.

Character extraction

Character extraction is one of the most useful image feature extraction techniques. One promising application is license plate reading. However, ordinary outdoor images under a wide variety of lighting conditions often include a strong shadow. Since a shadow forms edges across the characters, it is very difficult to extract characters from such images. Morphology based thresholding [77] is a promising technique for solving this problem.

Figure 5.12 is an example of character extraction from a license plate image with a strong shadow. The processing contains closing, subtraction and thresholding. As explained in Sections 5.3 and 5.4, CAM^2 can handle all of them efficiently. It takes about 0.24 msec to perform the character extraction.

Multiple object tracking

Another example of image processing using CAM^2 is shown in Fig. 5.13. By applying various morphological operations to perform line erasure, edge detection, hole filling and noise reduction, a binary image of target objects can be obtained. The processing requires 15 morphological operations with various structuring elements. CAM^2 can do it in just 200 μsec. As shown in Section 5.5.2, the data loading and retrieval times are 0.1 msec and 1.6 msec for 128 $\times$ 128 and 512 $\times$ 512 images, respectively. The processing can be finished at video rates.

Figure 5.14 shows an example of CAM^2_PL for the processing in Fig. 5.13. Here, "copy_8" means the intra-word copy of 8 bits and "sub_data8" means pixel-by-pixel subtraction of 8 bits. "Dilation" and "erosion" are the dedicated operations for describing morphological operations with various structuring elements, as mentioned before. Using

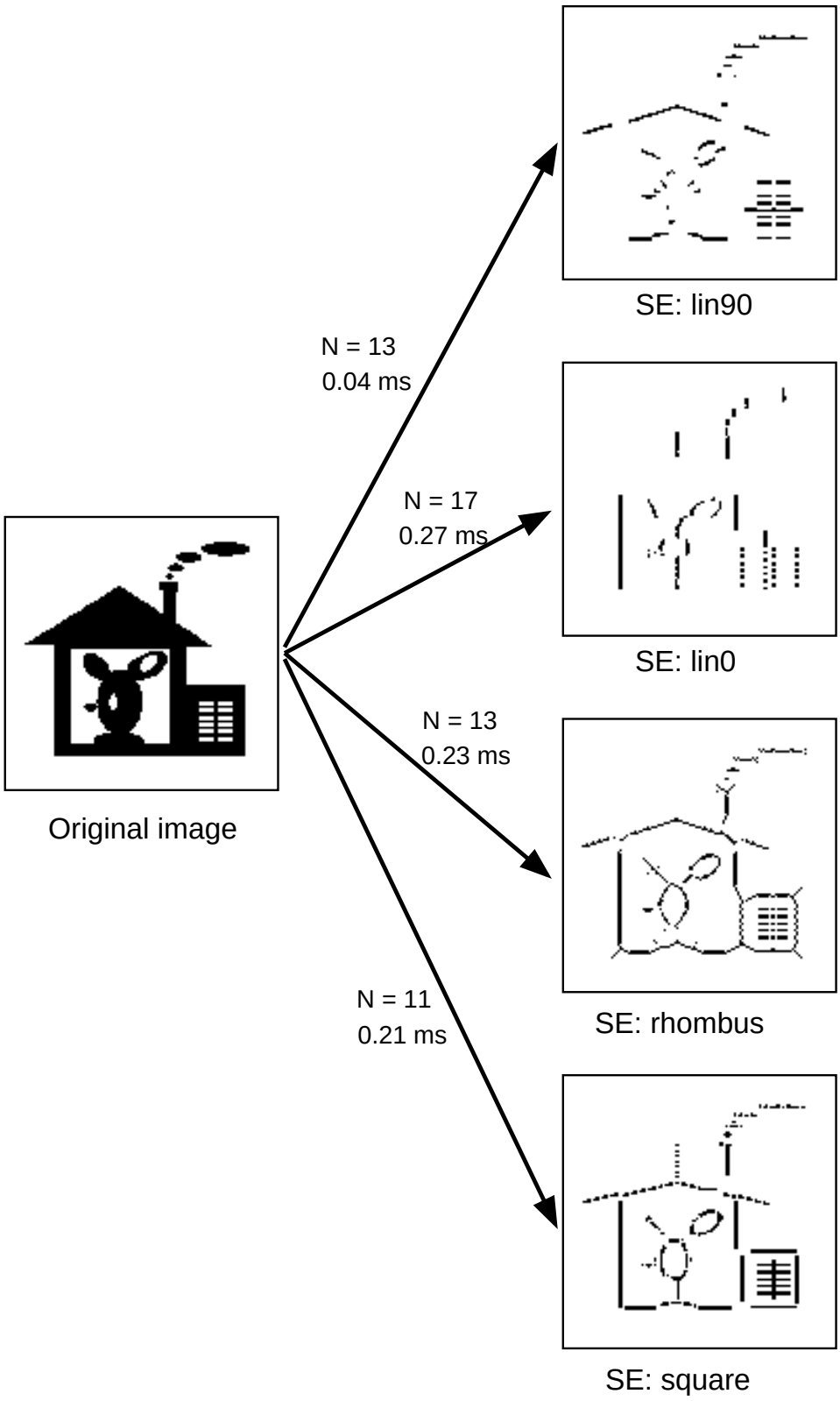


Figure 5.11: Morphological skeleton with various structuring elements.

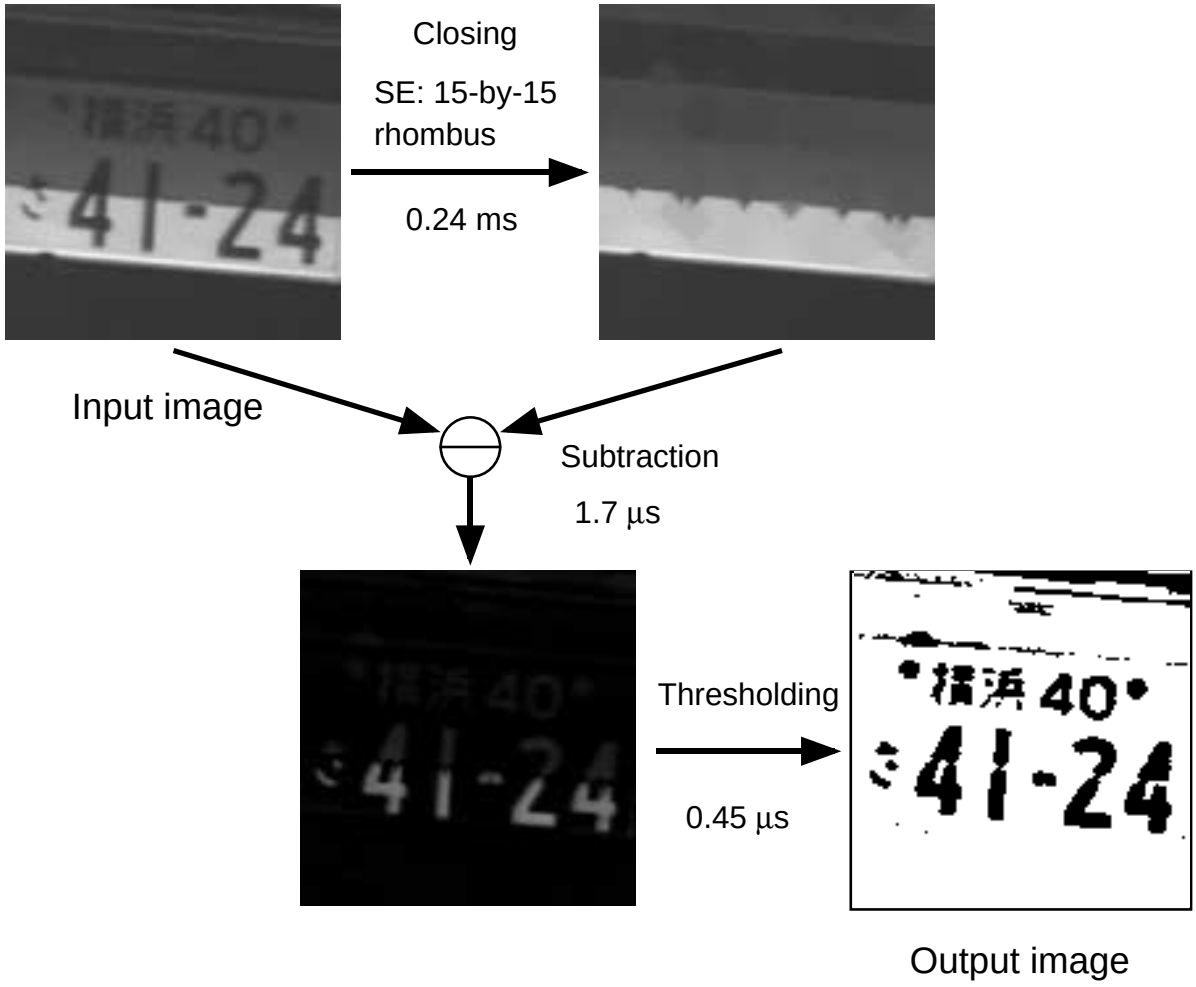


Figure 5.12: Character extraction for a license plate image with a strong shadow.

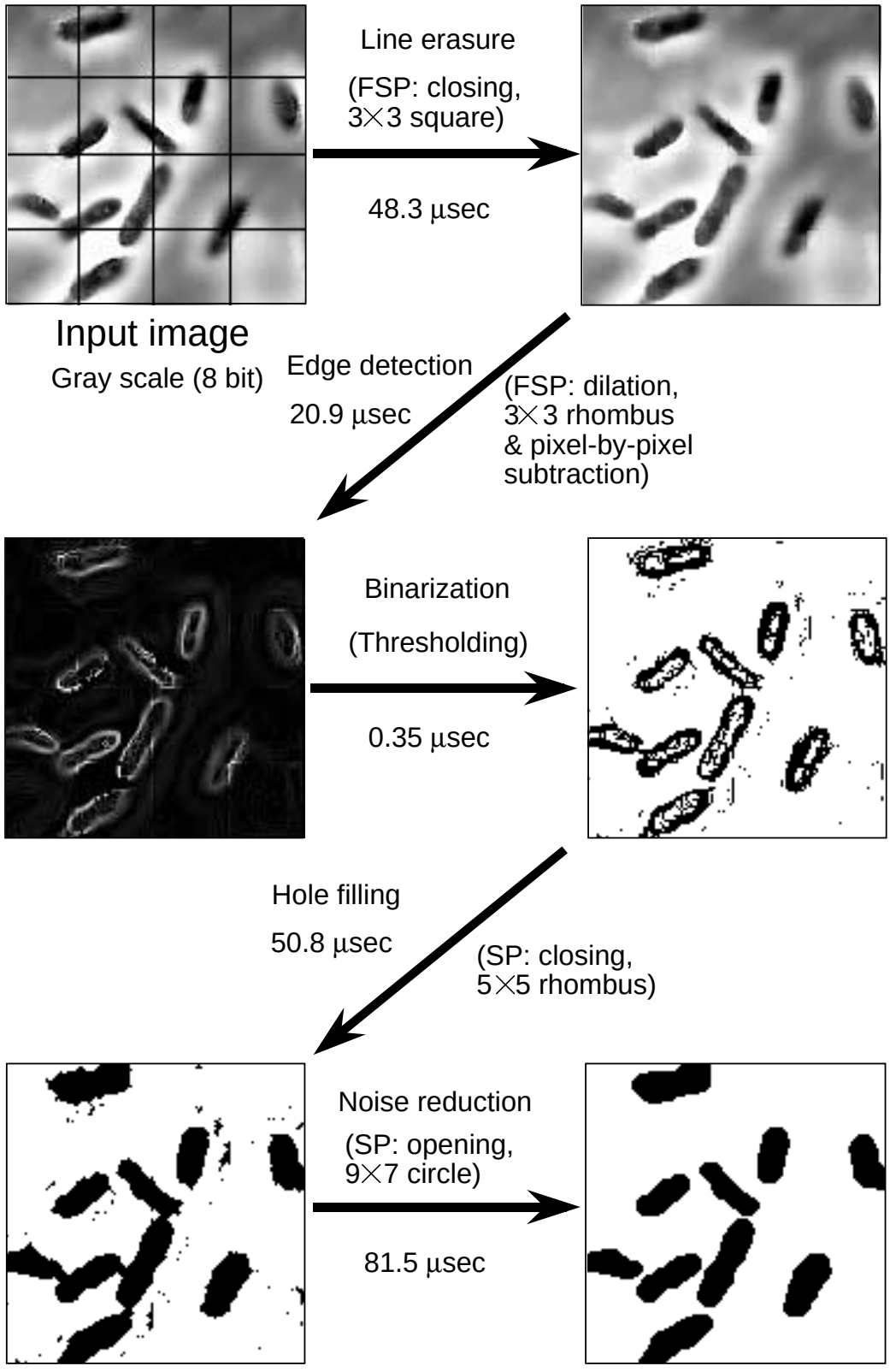


Figure 5.13: Multiple object tracking (morphology).

these operations, the processing is described in only 20 operations.

```
;; CAM^2_PL sample program (multiple object detection using morphology)

(gray_data_in)              ; Store gray scale image data
(copy_8 1 2)                ; Duplicate image data (fld1 -> fld2)
(dilation 'FSP 'square)     ; Line erasure (closing 3-by-3)
(erosion 'FSP 'square)
(dilation 'FSP 'rhombus)    ; Edge detection (dilation 3-by-3)
(sub_data8 1 2)             ; Pixel-by-pixel subtraction (fld1 - fld2 -> fld1)
(thresh 16)                 ; Binarization (threshold value = 16)
(repeat 2 '(                ; Hole filling (closing 5-by-5)
  (dilation 'SP 'rhombus)))
(repeat 2 '(
  (erosion 'SP 'rhombus)))
(erosion 'SP 'rhombus)      ; Noise reduction & Separation
(erosion 'SP 'square)       ; (opening 9-by-7)
(erosion 'SP 'rhombus)
(erosion 'SP 'lin0)
(dilation 'SP 'rhombus)
(dilation 'SP 'square)
(dilation 'SP 'rhombus)
(dilation 'SP 'lin0)
(bin_data_out)              ; Read binary image data
```

Figure 5.14: CAM^2_PL example for multiple object tracking.

As discussed above, CAM^2 efficiently performs not only morphology, but also other CA-based algorithms. Using these algorithms, the center points of target objects and a distance map for them can be obtained as shown in Fig. 5.15. By applying the processings in Figs. 5.13 and 5.15 to the input image and by finding the center points nearest those in the previous frame, multiple object tracking can be performed.

These examples demonstrate that CAM^2 is flexible enough to perform practical image processing employing a combination of morphology and other algorithms.

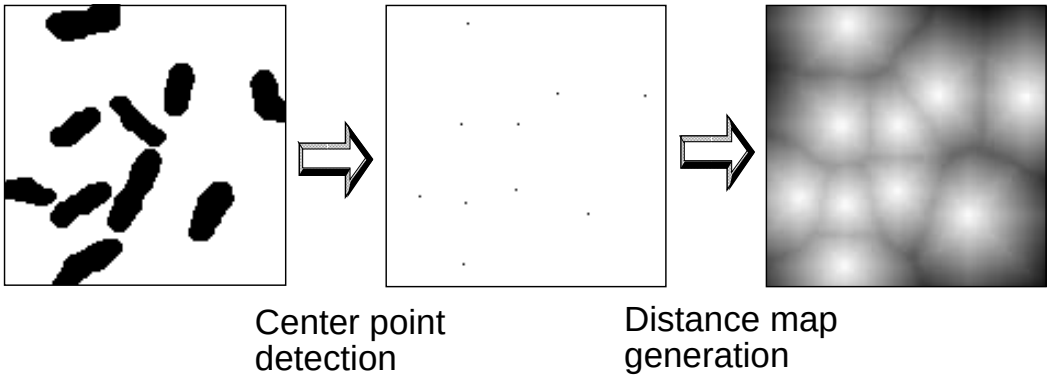


Figure 5.15: Multiple object tracking (other CA).

5.6 Conclusion

This chapter described a morphology processing method based on a highly-parallel two-dimensional cellular automata called CAM^2 and presented some evaluation results. New mapping methods using maskable search, partial & parallel write and hit-flag shift achieve high-throughput complex morphology processing.

Evaluation results show that CAM^2 performs one morphological operation for basic structuring elements within 30 μsec. This means that more than a thousand operations can be carried out on an entire 512 $\times$ 512 pixel image at video rates (33 msec). CAM^2 can also handle an extremely large and complex 100×100 structuring element at video rates. Furthermore, CAM^2 can perform practical image processing, such as pattern spectrum, skeletonization, character extraction and multiple object tracking, through a combination of morphology and other algorithms. Thus, CAM^2 will enable fuller realization of the potential of morphology and make a significant contribution to the development of real-time image processing systems based on morphology and other algorithms.

Chapter 6

Conclusion

6.1 Summary of results

This dissertation proposed a high-performance, compact, and flexible two-dimensional highly-parallel cellular automaton called CAM^2 for real-time image-understanding processing, which can be used to implement various image-understanding applications in the fields of industrial inspection, medical imaging, intelligent transportation systems, robotics, multimedia, human interface, entertainment, image coding, and so forth. To produce CAM^2 and demonstrate its usefulness for image-understanding processing, this dissertation covered three basic problem areas, namely computer architecture, LSI and system design and implementation, and applications.

As regards computer architecture, the study focused on the following three architectural considerations: CA mapping, CA processing, and data loading and retrieval processing. Multiple-zigzag mapping enables two-dimensional CA cells to be mapped into CAM words, even though physically a CAM has a one-dimensional structure. Dedicated CAM functions enable high-performance CA processing. Furthermore, parallel loading and partial word retrieval techniques enable high-throughput data transfer between CAM^2 and the outside image buffer. The performance evaluation results show that 256 k CA cells, which correspond to a 512×512 pixel picture, can be processed by a CAM^2 on a single board using deep sub-micron process technology. Moreover, the processing speed is more than 10 billion CA cell updates per second. This means that more than a thousand CA-

based image processing operations can be performed on a 512 × 512 pixel image at video rates (33 msec). Thus, the first conclusion to be drawn from this research is that CAM^2 will represent a major step toward the development of a compact and high-performance two-dimensional cellular automaton.

In terms of design and implementation, I fabricated a fully-parallel 1-Mb CAM LSI with dedicated functions for CA processing and a prototype CAM^2 PC board using this CAM chip. To satisfy the extremely severe design constraints of state-of-the-art process technology (0.25 μm), this study involves not only VLSI circuit design, but also packaging technology, circuit board fabrication technology, power and signal distribution techniques, heat dissipation problems and design and verification strategy. A CAM chip capable of operating at 56 MHz and 2.5 V was fabricated using 0.25-μm full-custom CMOS technology with five aluminum layers. A total of 15.5 million transistors were integrated into a 16.1 × 17.0 mm chip. The typical power dissipation was 0.25 W. The processing of various update and data transfer operations was performed at 3-640 GOPS. Since the fabricated CAM has 16 k words, or processing elements (PEs), which can process 128 × 128 pixels in parallel, a board-sized pixel-parallel image processing system can be implemented using several chips. Indeed, the prototype board has a two-dimensional array (2 × 2) of CAM chips, and can handle a 256 × 256 pixel image. Since PCI bus and NTSC video interfaces are also embedded in the board, a compact image-processing platform can be built simply by connecting the board to a personal computer and a video camera. Thus, the second conclusion of this research is that an economically feasible, compact, high-performance, and flexible CAM^2 system can actually be obtained with the current technology.

As regards application, I focused on two fairly advanced CA-based computation paradigms, namely discrete-time cellular neural network (DTCNN) and mathematical morphology. DTCNN is a promising computer paradigm that fuses artificial neural networks with the concept of the cellular automaton (CA). Mathematical morphology is an image transformation technique that locally modifies geometric features through set operations. Both are powerful tools with various applications in image processing field. Here, I studied

mapping and processing method needed to perform various kinds of DTCNN and morphology processing. Evaluation results show that, on average, CAM^2 can perform one transition for various DTCNN templates in about 12 μsec. CAM^2 also can perform one morphological operation for basic structuring elements within 30 μsec. These results mean that more than a thousand operations can be carried out on an entire 512 $\times$ 512 pixel image at video rates (33 msec). Furthermore, CAM^2 can perform practical image processing through a combination of DTCNN, morphology, and other algorithms. Thus, the last conclusion is that CAM^2 will enable fuller realization of the potential of DTCNN and morphology and make a significant contribution to the development of real-time image processing systems based on DTCNN and morphology and a combination of them.

6.2 Future work

Although this dissertation has solved the problem of designing a highly-parallel computer for real-time image-understanding processing that is extremely compact and that provides very high levels of performance, much remains to be done in terms of further improving the performance and expanding the use to various vision applications. Below, I mention three aspects related to the refinement of CAM^2 that I believe to be important.

6.2.1 Off-chip high-speed memory

Since each cell of the current CAM^2 system has a 64-bit storage capacity, the system can store eight 8-bit gray-scale image planes simultaneously. Although this capacity is not very large, CAM^2 can handle a wide variety of vision applications as described in Chapters 4 and 5. However, there are certain applications, such as color-based applications or applications that must store past frame data for a long period of time, that require a large capacity for storing various temporary data. Although the capacity of CAM^2 will increase markedly with progress on LSI process technologies, occasionally it would be insufficient for the above applications. Moreover, considering the cost issue, the capacity should be modified adaptively according to the demand for applications.

A promising solution to this problem involves using an off-chip memory based on high-speed electrical signaling technology [85], such as Rambus DRAM. Rambus DRAM (RDRAM) [86] is a general purpose high-performance memory device suitable for use in a broad range of applications including computer memory, graphics, video, and any other application requiring high bandwidth and low latency. For example, the 128/144-Mbit Direct Rambus DRAMs permit 600 to 800 MHz transfer rates while using conventional system and board design technologies. They are capable of sustained data transfers at 1.25 ns per two bytes. A rough evaluation shows that the capacity of the current CAM^2 system increases to 4 Gbit, which means 8000 image planes can be stored simultaneously, simply by combining the commercially available Rambus DRAMs with maximum capacity and CAM^2. Moreover, a Rambus I/F module can be embedded in a CAM chip with an acceptable increase in the required amount of hardware and number of I/O pins. Furthermore, by executing off-chip data transfer operations during the existing operation cycles, such as the hit-flag shift, on-chip data can be transferred to off-chip memory and vice versa without wasting very much processing time. Thus, the combination of CAM^2 and off-chip high-speed memory is an easy but effective way to broader the vision application range.

6.2.2 Multilevel vision architecture

CAM^2 can handle most low-level and some intermediate-level vision tasks efficiently as shown in Chapters 4 and 5. However, for whole vision processing we must find a way of dealing with the intermediate-level vision task that is beyond the scope of CAM^2 and the high-level vision task. With the HiPIC concept, these areas of processing are assumed to be executed using a DSP or a host computer (microprocessor). However, since some algorithms for real-time applications also require a huge amount of computational power, an accelerator that covers both level tasks is also desired.

The most suitable architectures for these three levels are different because the algorithms for each level have different network structures and degrees of parallelism. So,

a multilevel architecture, such as image-understanding architecture (IUA) [87, 88], is a good approach to solving the problem. IUA consists of three different, tightly coupled parallel processors: the content addressable array parallel processor (CAAPP) at the low level, the intermediate communications associative processor (ICAP) at the intermediate level, and the symbolic processing array (SPA) at the high level. Although IUA offer an ideal environment for vision applications, its drawback is the huge amount of hardware it requires. So, to create a compact multilevel vision system that is widely used, we must study architectural considerations based on LSI technology.

The architectural candidates that I consider promising for each level are as follows:

- Low level: CAM^2 or CAM^2 with off-chip memory (Since the functions of CAAPP and CAM^2 are very similar, CAAPP can be easily replaced by CAM^2.)

- Intermediate level: Versatile linear array (Although there are no actual LSI chips that can be substituted for the ICAP, some architectural concepts, such as asynchronous SIMD (ASIMD) [89], have been proposed. ASIMD is a kind of linear array architecture but has some advanced functions such as a virtualization manager and a data formatter, which offer greater flexibility for intermediate-level vision tasks.)

- High level: MIMD type DSP (Because of the huge progress made in LSI technology, a general-purpose MIMD DSP, such as [90], has both high performance and sufficient flexibility for high-level vision tasks. It can be substituted for the SPA.)

All of the above can be implemented in one or a few chips, and the whole system can be embedded as a multi-chip module or in a board. In the near future, they will be able to be embedded in a chip. This will enable us to obtain a compact multilevel vision system that covers all vision applications. This, in turn, will offer greater computer vision applicability.

6.2.3 Humanoid computer with multi-modal man-machine interface

Computers have been used for more than a half a century and are becoming indispensable in various aspects of life. However, the interface (I/F) is still based on such legacy devices as a keyboard and a mouse. Moreover, we still have to master special skills such as touch typing to use computers efficiently. So, a computer with a user-friendly I/F is highly desirable.

A humanoid computer with a multi-modal man-machine I/F is an ultimate goal. A computer that acted as if it were human might offer the most natural means of interaction between humans and machines. Speech and vision processing would play a particularly important role when creating such a computer. Research is now under way on a natural dialog system [91], [92] with a speech and vision I/F.

With the recent rapid progress in microprocessor technology, most speech recognition and synthesis algorithms can be performed by a general-purpose microprocessor in real-time. However, it is still very difficult to execute image recognition and synthesis processing on a microprocessor. Indeed, the current PC consists of not only a microprocessor but also a graphic accelerator chip to perform image or graphic synthesis efficiently. In contrast, there are no good compact platforms for handling image recognition. The multilevel vision architecture mentioned above would be a good solution. Figure 6.1 shows the concept of a humanoid computer that covers both image recognition and synthesis capability.

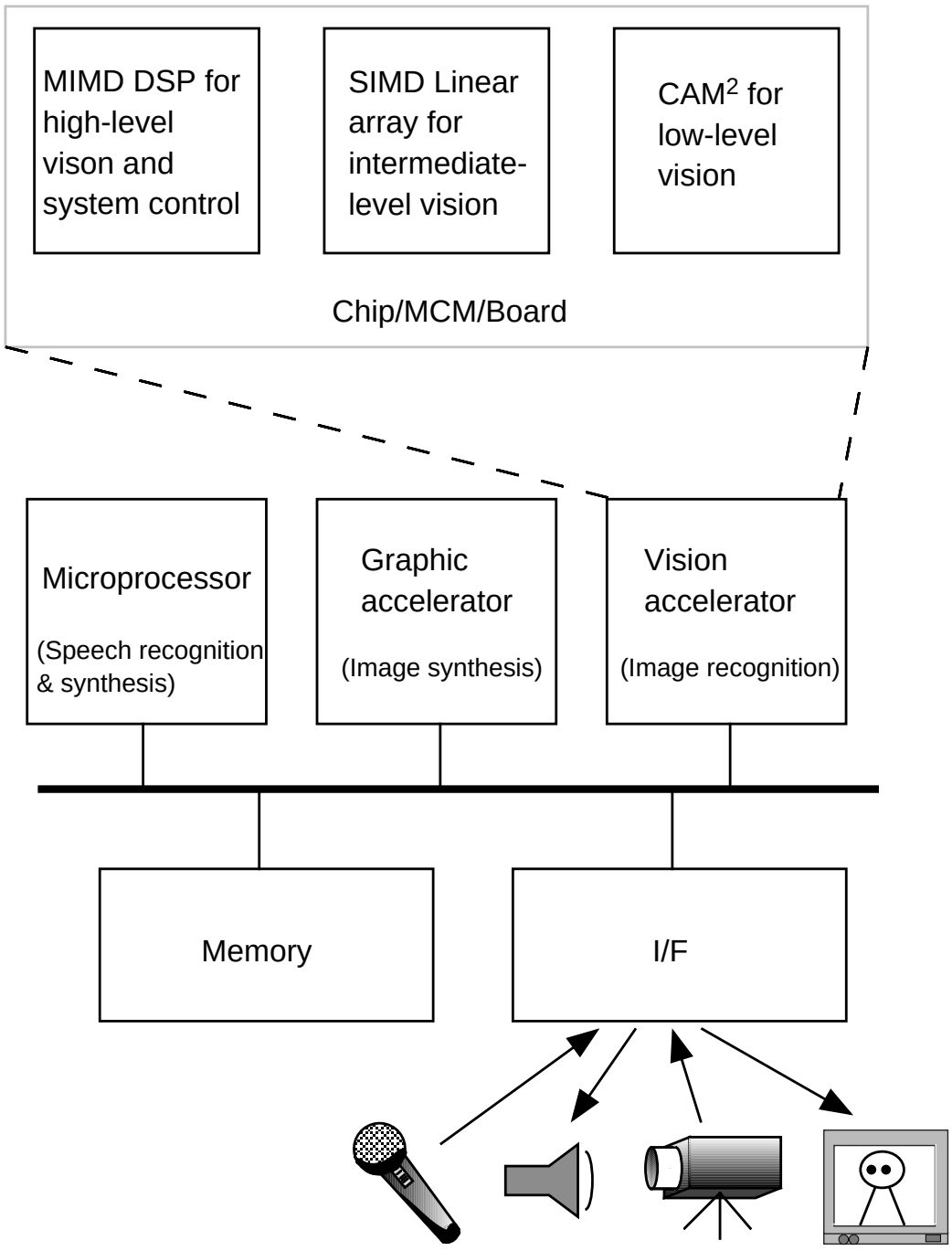


Figure 6.1: Concept of a humanoid computer with multi-modal man-machine I/F.

Bibliography

[1] J. von Neumann, "The General Logical Theory of Automata," Cerebral Mechanisms in Behavior - The Hixon Symposium, L. A. Jeffress, ed., John Wiley & Sons, New York, 1951.

[2] K. Preston, Jr. and M. J. B. Duff, "Modern Cellular Automata," Plenum Press, New York and London, 1984.

[3] M. J. Flynn, "Very High Speed Computing Systems," *Proc. of the IEEE*, Vol. 54, pp. 1901–1909, 1966.

[4] R. Duncan, "A Survey of Parallel Computer Architectures," *IEEE Computer*, pp. 5–16, 1990.

[5] M. Sipper, "Evolution of Parallel Cellular Machines," Springer-Verlag, Verlin Heidelverg, 1997.

[6] K. Preston, Jr., et al., "Basics of Cellular Logic with Some Applications in Medical Image Processing," *Proc. of the IEEE*, vol. 67, no. 5, pp. 826–857, 1979.

[7] E. R. Dougherty and P. A. Laplante, "Real-time Imaging," SPIE & IEEE Press, Washington and New York, 1995.

[8] P. A. Laplante and A. D. Stoyenko, "Real-time Imaging: Theory, Techniques, and Applications," IEEE Press, New York, 1996.

[9] H. Harrer, "Multiple Layer Discrete-time Cellular Neural Networks using Time-

variant Templates," *IEEE Trans. Circuits Syst., II*, vol. 40, no. 3, pp. 191–199, 1993.

[10] H. Harrer, J. A. Nossek, T. Roska and L. O. Chua, "A Current-mode DTCNN Universal Chip," *Proc. IEEE Int'l Symp. Circuits Syst. (ISCAS'94)*, vol. 4, pp. 135–138, 1994.

[11] J. Serra, "Image Analysis and Mathematical Morphology," Academic Press, London, 1982.

[12] P. Maragos, "Tutorial on Advances in Morphological Image Processing and Analysis," *Opt. Eng.*, 26, 1987.

[13] C. C. Weems, "Architectural Requirements of Image Understanding with Respect to Parallel Processing," *Proc. of the IEEE*, vol. 79, no. 4, pp. 537–547, 1991.

[14] K. Preston Jr., "Cellular Logic Computers for Pattern Recognition," *IEEE Computer*, vol. 16, no. 1, pp. 36–47, 1983.

[15] M. J. B. Duff, et al., "Review of the CLIP Image Processing System," *Proc. of National Computer Conference*, pp. 1055–1060, 1978.

[16] K. E. Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. Computers*, vol. C-29, no. 9, pp. 836–840, 1980.

[17] T. Kondo, et al., "Pseudo MIMD Array Processor-AAP2," *Proc. of 13th Symposium on Computer Architecture Conf.*, pp. 330–337, 1986.

[18] Thinking Machines Corp., "Connection Machine Model CM-2 Technical Summary," ver. 5.1, 1989.

[19] J. R. Nickolls, "The Design of the Maspar MP-1: A Cost-effective Massively Parallel Computer," *Proc. of COMPCON Spring'90*, pp. 25–28, 1990.

[20] S. R. Sternberg, "Biomedical Image Processing," *IEEE Computer*, vol. 16, no. 1, pp. 22–34, 1983.

[21] T. Toffoli and N. Margolus, "Cellular Automata Machines," MIT Press, Cambridge, MA, 1987.

[22] T. Ogura and M. Nakanishi, "CAM-based Highly-parallel Image Processing Hardware," *IEICE Trans. Electron*, vol. E80-C, no. 7, pp. 868–874, 1997.

[23] Y. Fujino, T. Ogura and T. Tsuchiya, "Facial Image Tracking System Architecture utilizing Real-time Labeling," *Proc. SPIE Visual Comm. and Image Processing (VCIP'93)*, 1993.

[24] M. Nakanishi and T. Ogura, "A Real-time CAM-based Hough Transform Algorithm and its Performance Evaluation," *13th Int'l Conf. Pattern Recognition (ICPR'96)*, vol. 2, pp. 516–521, 1996.

[25] M. Nakanishi and T. Ogura, "Real-time Extraction using a Highly Parallel Hough Transform Board," *Proc. IEEE Int. conf. Image Processing (ICIP-97)*, vol. 2, pp. 582–585, 1997.

[26] M. Meribout, T. Ogura and M. Nakanishi, "Real-time Hough Transform based Circular Shape Extraction," *Proc. IAPR Workshop on Machine Vision Applications (MVA'96)*, pp. 178–182, 1996.

[27] M. Meribout, M. Nakanishi and T. Ogura, "Hough Transform Implementation on a Reconfigurable Highly Parallel Architecture," *Proc. Computer Architectures for Machine Perception (CAMP'97)*, pp. 276–279, 1997.

[28] E. Hosoya, T. Ogura and M. Nakanishi, "Real-time 3D Feature Extraction Hardware Algorithm with Feature Point Matching Capability," *Proc. IAPR Workshop on Machine Vision Applications (MVA'96)*, pp. 430–433, 1996.

[29] P. Kalapathy, et al., "Hardware/software Interaction on the Mpact Media Processor," *Hot Chips VIII symposium record*, pp. 171–178, 1996.

[30] T. Yoshida, et al., "A 2V 250MHz Multimedia Processor," *IEEE Int. Solid-State Circuits Conf. (ISSCC97), Digest of technical papers*, pp. 266–267, 1997.

[31] M. R. Choudhury, et al., "A 300MHz CMOS Microprocessor with Multi-media Technology," *IEEE Int. Solid-State Circuits Conf. (ISSCC97), Digest of technical papers*, pp. 170–171, 1997.

[32] D. A. Draper, el al., "An X86 Microprocessor with Multimedia Extensions," *IEEE Int. Solid-State Circuits Conf. (ISSCC97), Digest of technical papers*, pp. 172–173, 1997.

[33] N. Yamashita, et al., "A 3.84 GIPS Integrated Memory Array Processor with 64 Processing Elements and a 2-Mb SRAM," *IEEE J. Solid-State Circuits*, vol. 29, no. 11, pp. 1336–1343, 1994.

[34] M. Kurokawa, et al., "5.4GOPS Linear Array Architecture DSP for Video-Format Conversion," *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC96), Digest of technical papers*, pp. 254–255, 1996.

[35] R. Etienne-Cummings, et al., "A Foveated Visual Tracking Chip," *IEEE Int. Solid-State Circuits Conf. (ISSCC97), Digest of technical papers*, pp. 38–39, 1997.

[36] K. Aizawa, et al., "Implementations of on Sensor Image Compression and Comparisons Between Pixel and Column Parallel Architectures," *Proc. IEEE Int. conf. Image Processing (ICIP-97)*, vol. 2, pp. 258–261, 1997.

[37] T. Komuro, I. Ishii and M. Ishikawa, "Vision Chip Architecture using General-purpose Processing Elements for 1ms Vision System," *Proc. Computer Architectures for Machine Perception (CAMP'97)*, pp. 276–279, 1997.

[38] G. Estrin and R. H. Fuller, "Some Applications for Content Addressable Memory," *Proc. of FJCC*, pp. 495–508, 1963.

[39] B. T. McKeever, "Associative Memory Structure," *Proc. of FJCC*, pp. 371–388, 1965.

[40] S. S. Yau and H. S. Fung, "Associative Processor Architecture - A Survey," *Computing Surveys*, vol. 9, no. 1, pp. 3–27, 1977.

[41] A. Krikelis and C. C. Weems, "Associative Processing and Processors, " IEEE Computer Society Press, Los Alamitos, 1997.

[42] T. Ogura, S. Yamada and T. Nikaido, "A 4-kbit Associative Memory LSI," *IEEE J. Solid-State Circuits*, vol. SC-20, no. 6, pp. 1277–1282, 1985.

[43] H. Kadota, J. Miyake, Y. Nishimichi, H. Kudoh and K. Kagawa, "An 8-kbit Content-addressable and Reentrant Memory," *IEEE J. Solid-State Circuits*, vol. SC-20, no. 6, pp. 951–963, 1985.

[44] H. Yamada, et al., "Real-time String Search Engine LSI for 800-Mbit/sec LANs," *Proc. IEEE Custom Integrated Circuits Conf. (CICC'88)*, pp. 21.6.1–21.6.4, 1988.

[45] T. Ogura, et al., "A 20-kbit Associative Memory LSI for Artificial Intelligence machines," *IEEE J. Solid-State Circuits*, vol. 24, no. 4, pp. 1014–1020, 1989.

[46] T. Yamagata, et al., "A 288-kb Fully Parallel Content Addressable Memory using a Stacked-capacitor Cell Structure," *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1927–1933, 1992.

[47] S. Ushijima, et al., "High-performance Connectionless Server for Public ATM Networks and its Application to Multiple Services Support in Heterogeneous Networking Environments," *Proc. 2nd Int'l Symp. on Interworking*, 1994.

[48] T. Ogura, M. Nakanishi, T. Baba, Y. Nakabayashi and R. Kasai, "A 336-kbit Content Addressable Memory for Highly Parallel Image Processing," *Proc. IEEE Custom Integrated Circuits Conf. (CICC'96)*, vol. 13. 4. 1, pp. 273–276, 1996.

[49] F. Shafai, et al., "Fully Parallel 30-MHz, 2.5-Mb CAM," *IEEE J. Solid-State Circuits*, vol. 33, no. 11, pp. 1690-1696, 1998.

[50] C. C. Foster, "Content Addressable Parallel Processors," Van Nostrand Reinhold, New York, 1976.

[51] C. C. Weems, E. M. Riseman and A. R. Hanson, "Image Understanding Architecture: Exploiting Potential Parallelism in Machine Vision," *IEEE Computer*, pp. 65–68, 1992

[52] J. C. Gealow and C. G. Sodini, "A Pixel-Parallel Image Processor Using Logic Pitch-Matched to Dynamic Memory, " *IEEE J. Solid-State Circuits*, vol. 34, no. 6, pp. 831–839, 1999

[53] M. Sonka, V. Hlavac and R. Boyle, "Image Processing, Analysis and Machine Vision," Chapman & Hall, London, 1993.

[54] J. C. Russ, "The Image Processing Handbook," CRC Press Inc., Boca Raton, 1995.

[55] E. Sternheim, et al., "Digital design with Verilog HDL," Automata Publishing Company, 1990.

[56] "The Programmable Logic Data Book 1998," Xilinx Inc., San Jose, 1998.

[57] "PLX New Products Catalog," PLX Technology, Sunnyvale, 1996.

[58] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1257–1272, 1988.

[59] L. O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE Trans. Circuits Syst.*, vol. 35, pp. 1273–1290, 1988.

[60] T. Roska and L. O. Chua, "The CNN Universal Machine: An Analogic Array Computer," *IEEE Trans. Circuits Syst., II*, vol. 40, pp. 163–173, 1993.

[61] H. Saito, K. Jin-no and M. Tanaka, "Universality of DT-CNN," *Technical report of IEICE*, CAS96-34, NLP96-72, pp. 65–72, 1996 (in Japanese).

[62] T. Matsumoto, L. O. Chua and R. Furukawa, "CNN Cloning Template: Hole-filler," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 635–638, 1990.

[63] T. Matsumoto, L. O. Chua and H. Suzuki, "CNN Cloning Template: Connected Component Detector," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 633–635, 1990.

[64] T. Matsumoto, L. O. Chua and T. Yokohama, "Image Thinning with a Cellular Neural Network," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 638–640, 1990.

[65] H. Harrer, J. A. Nossek, T. Roska and L. O. Chua, "A Current-mode DTCNN Universal Chip," *Proc. IEEE Int'l Symp. Circuits Syst. (ISCAS'94)*, vol. 4, pp. 135–138, 1994.

[66] H. Numata and M. Tanaka, "Design of Universal Pipelining Discrete Time Cellular Neural Network by MOSIS," *Proc. Int'l Symp. Nonlinear Theory and its Applications (NOLTA'95)*, vol. 2, pp. 655–660, 1995.

[67] A. Rodriguez Vázquez, S. Espejo and R. Dominguez-Castro, "Design Considerations for High-performance CNN Universal Chips," *Proc. Int'l Symp. Nonlinear Theory and its Applications (NOLTA'95)*, vol. 2, pp. 649–654, 1995.

[68] J. M. Cruz and L. O. Chua, "A Core Cell and I/O Circuitry of an Analog-input Dual-output CNN Universal Chip," *Proc. Int'l Symp. Nonlinear Theory and its Applications (NOLTA'95)*, vol. 2, pp. 661–666, 1995.

[69] T. Matsumoto, L. O. Chua and H. Suzuki, "CNN Cloning Template: Shadow Detector," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 1070–1073, 1990.

[70] T. Roska, T. Boros, P. Thiran, L. O. Chua, "Detecting Simple Motion Using Cellular Neural Networks," *Proc. IEEE Int. Workshop on Cellular Neural Networks and their applications (CNNA-90)*, pp. 127–139, 1990.

[71] T. Roska, L. O. Chua, "Cellular Neural Networks with Nonlinear and Delaytype Templates Elements," *Proc. IEEE Int. Workshop on Cellular Neural Networks and their applications (CNNA-90)*, pp. 12–25, 1990.

[72] G. Matheron, "Random Sets and Integral Geometry," Wiley, New York, 1975.

[73] H. Kobatake, "Morphology," Corona Publishing Co., Tokyo, 1996 (in Japanese).

[74] R. M. Haralick, S. R. Sternberg and X. Zhuang, "Image Analysis using Mathematical Morphology," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp.532–550, 1987.

[75] L. Vincent, "Graphs and Mathematical Morphology," *Signal Processing*, vol. 16, no. 4, pp.365–388, 1989.

[76] S. Yamamoto, M. Matsumoto, Y. Tateno, T. Iinuma and T. Matsumoto, "Quoit filter — a New Filter based on Mathematical Morphology to Extract the Isolated Shadow, and its Application to Automatic Detection of Lung Cancer in X-ray CT," *13th Int'l Conf. Pattern Recognition (ICPR'96)*, Vol.2, pp.3–7, 1996.

[77] Y. Takahashi, A. Shio and K. Ishii, "Morphology based Thresholding for Character Extraction," *IEICE Trans. Inf. & Syst.*, vol. E76-D, no. 10, pp.1208–1215, 1997.

[78] M. Hassoun, T. Meyer, P. Siqueira, J. Basart and S. Gopalratnam, "A VLSI Gray-scale Morphology Processor for Real-time NDE Image Processing Applications," *SPIE Vol. 1350 Image Algebra and Morphological Image Processing*, 1990.

[79] R. Lin and E. K. Wong, "Logic Gate Implementation for Gray-scale Morphology," *Pattern Recognition Letters*, vol. 13, no. 7, 1992.

[80] C. H. Chen and D. L. Yang, "Realization of Morphological Operations," *IEE Proc. Circuits Devices Syst.*, vol. 142, 1995.

[81] L. Lucke and C. Chakrabarti, "A digital-serial Architecture for Gray-scale Morphological Filtering," *IEEE Trans. Image Processing*, vol. 4, no. 3, pp. 387–391, 1995.

[82] E. R. Dougherty and D. Sinha, "Computational Gray-scale Mathematical Morphology on Lattices (A Comparator-based Image Algebra) Part I: Architecture," *Real-time Imaging*, vol. 1, pp. 69–85, 1995.

[83] P. Maragos, "Pattern Spectrum and Multiscale Shape Representation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol . 11, no. 7, pp. 701–716, 1989.

[84] K. Sudo, J. Yamato and A. Tomono, "Determining Gender using Morphological Pattern Spectrum," *IEICE Trans. Inf. & Syst.*, vol. J80-D-II no.5 pp.1037–1045, 1997 (in Japanese).

[85] M. Horowitz, C. K. Yang and S. Sidiropoulos, "High-speed Electrical Signaling: Overview and Limitations," *IEEE Micro*, vol. 18, no. 1, pp. 12–24, 1998.

[86] "Direct RDRAM 128/144-Mbit," RAMBUS Inc., 2000.

[87] C. C. Weems, S. P. Levitan, A. R. Hanson and E. M. Riseman, "Image Understanding Architecture," *Int'l Journal of Computer Vision*, pp. 251–282, 1989.

[88] C. C. Weems, E. M. Riseman and A. R. Hanson, "Image Understanding Architecture: Exploiting Potential Parallelism in Machine Vision," *IEEE Computer*, pp. 65–68, 1992.

[89] C. C. Weems, "Asynchronous SIMD: An Architectural Concept for High Performance Image Processing," *Proc. Computer Architectures for Machine Perception (CAMP'97)*, pp. 235–242, 1997.

[90] B. Ackland, et al., "A Single-chip, 1.6-billion, 16-b MAC/s Multiprocessor DSP," *IEEE J. Solid-State Circuits*, vol. 35, no. 3, pp. 412–424, 2000.

[91] T. Kobayashi and K. Shirai, "Multi-modal Conversational Interface for Humanoid Robot," *Technical report of IEICE*, SP99-113, pp. 121– 126, 1999 (in Japanese).

[92] M. Nakano, K. Dohsaka, N. Miyazaki, J. Hirasawa, M. Tamoto, M. Kawaori. A. Sugiyama, and T. Kawabata, "A Real-time Spoken Dialog System for TV-Program Recording Reservations," *SIG-SLP-22, Information Processing Society of Japan*, pp. 41-42, 1998 (in Japanese).

Author Biography

Takeshi Ikenaga was born in Fukuoka Prefecture, Japan on July 8, 1964. He received B.E. and M.E. degrees in electrical engineering from Waseda University, Tokyo, Japan, in 1988, and 1990, respectively, where he belonged to the Information Systems Laboratory directed by Professor Katsuhiko Shirai.

He joined LSI Laboratories, Nippon Telegraph and Telephone Corporation (NTT) in 1990, where he has been undertaking research on the design and test methodologies for high-performance ASICs. From 1992 to 1995, he worked at the Signal Processor LSI Research Group in NTT LSI Laboratories, where he was mainly engaged in the development of a real-time MPEG2 encoder chip set. In 1996, he moved to the Highly Parallel Processing Research Group in NTT System Electronics Laboratories, where he was engaged in research on the highly parallel LSI & system design for image-understanding processing (the scope of this dissertation). From 1999 to 2000, he was a visiting researcher at the Architecture & Language Implementation (ALI) Group (directed by Professor Charles C. Weems) of the Department of Computer Science, University of Massachusetts, Amherst, USA.

He is presently a senior research engineer with the Home Communication Research Group of NTT Lifestyle and Environmental Technology Laboratories. His current interests include highly parallel LSI & system design and its application to computer vision, humanoid systems with multi-modal interfaces, and home communication and network systems.

He is a member of the Institute of Electrical and Electronics Engineers (IEEE), the Institute of Electronics, Information and Communication Engineers of Japan (IEICE),

and the Information Processing Society of Japan (IPSJ).

Between 1987 to 1990, he was the recipient of the Okuma Memorial Fellowship awarded by Waseda University. He also received the Furukawa Sansui award from Waseda University in 1988. In 1992, he received the IEICE Research Encouragement Award for his paper "A Test Pattern Generation for Arithmetic Execution Units".

Publication List

Journal Papers

1. ◯ [1] T. Ikenaga and T. Ogura, "Real-time Morphology Processing using Highly-parallel 2D Cellular Automata CAM^2," *IEEE Trans. Image Processing*, vol. 9, no. 12, pp. 2018–2026, Dec. 2000. {The contents of Chapter 5}

2. ◯ T. Ikenaga and T. Ogura, "A Fully Parallel 1-Mb CAM LSI for Real-time Pixel-parallel Image Processing," *IEEE J. Solid-state Circuits*, vol. 35, no. 4, pp. 536–544, Apr. 2000. {The contents of Chapter 3}

3. ◯ T. Ikenaga and T. Ogura, "CAM^2: A Highly-parallel 2D Cellular Automata Architecture," *IEEE Trans. Comput.*, vol. 47, no. 7, pp. 788–801, Jul. 1998. {The contents of Chapter 2}

4. ◯ T. Ikenaga and T. Ogura, "A DTCNN Universal Machine based on Highly-parallel 2-D Cellular Automata CAM^2," *IEEE Trans. Circuits Syst. I*, vol. 45, no. 5, pp. 538–546, May 1998. {The contents of Chapter 4}

5. T. Ikenaga and T. Ogura, "A Distributed BIST Technique and Its Test Design Platform for VLSIs," *IEICE Trans. Electronics*, vol. E78-C, no. 11, pp. 1618–1623, Nov. 1995

6. T. Ikenaga and J. Takahashi, "A Built-In Self-Test Structure for Arithmetic Execution Units of VLSIs," *IEICE Trans. Electronics*, vol. J77-C-II, no. 10, pp. 419–427, Oct. 1994 (in Japanese)

[1] ◯: papers related to this dissertation

7. T. Ikenaga and K. Shirai, "Special Purpose Processor Design System Realizing Algorithm Described by Higher Level Language," *IPSJ Trans.*, vol. 32, no. 11, pp. 1445–1456, Nov. 1991 (in Japanese)

International Conference Papers

1. ◯ T. Ikenaga and T. Ogura, "CAM^2: Content Addressable Memory LSI for 2D Cellular Automata Machine," *Proc. Int'l Workshop on Advanced LSI's and Devices*, pp. 197–204, Jul. 1999

2. ◯ T. Ikenaga and T. Ogura, "A Fully-parallel 1Mb CAM LSI for Real-time Pixel-parallel Image Processing," *IEEE Int'l Solid-State Circuit conf. (ISSCC99), Digest of technical papers*, 15-5, pp. 264–265, Feb. 1999

3. ◯ T. Ikenaga and T. Ogura, "Real-time Morphology Processing using Highly-parallel 2D Cellular Automata CAM^2," *Proc. IEEE Int'l conf. Image Processing (ICIP-97)*, TP04.1, vol. 2, pp. 562–565, Nov. 1997

4. ◯ T. Ikenaga and T. Ogura, "Discrete-time Cellular Neural Networks using Highly-parallel 2D Cellular Automata CAM^2," *Proc. Int'l Symp. Nonlinear Theory and its Applications (NOLTA '96)*, pp. 221–224, Oct. 1996

5. ◯ T. Ikenaga and T. Ogura, "CAM^2: A Highly-parallel 2-D Cellular Automata Architecture for Real-time and Palm-top Pixel-level Image Processing," *Springer LNCS 1124: Euro-Par'96*, vol. 2, pp.203–666, Aug. 1996

6. T. Ikenaga and T. Ogura, "A Test Design Platform for Shorter TAT Implementation of a Distributed BIST Structure," *Proc. European Design and Test Conf. (ED&TC95)*, User's Forum, pp. 215–218, Mar. 1995