Generic Associative Memory for Information Retrieval

C.H. Ben CHOI
Dept. of Electrical Engineering

Dik L. LEE
Dept. of Computer & Information Science

The Ohio State University
Columbus, Ohio 43210

ABSTRACT

The design of a general purpose associative memory is described. Its applications on Prolog clause indexing and on Relational database search/retrieval operations are explored. This associative memory can perform parallel search for multi-word strings or records, and then retrieve any number of pointers or data. It can be used as a standard RAM for random read/write operations, and as a conventional associative memory for relational search and pattern matching.

I. Introduction

Associative Memory, alternatively known as Content Addressable Memory, differs from regular memory in the way the stored data are retrieved. Data stored in associative memory are addressed by their contents, not by their physical locations. The major advantage of associative memory over RAM is its ability to perform parallel search and pattern matching. This ability makes associative memory preferable in many applications, such as image processing [1], database system [2], and Artificial Intelligent computers [3; 4; 5; 6].

A conventional associative memory is structured in the form of an array (see Fig. 1). Each word has its own comparison circuit and a tag bit which is used to store the result of the comparison corresponding to that word. In a search operation, all the words in the associative

Research is supported by NSF Grant IRI-881064.
Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

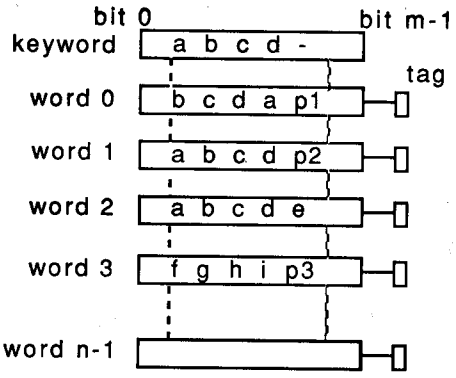

Fig. 1. A conventional associative memory.

memory are compared with a keyword simultaneously. And, the results are stored in the corresponding tag bits. The tag bits indicate which word(s) matches the keyword. The matched words can be read sequentially using a multiple response resolver [2; 7].

Although words and bits of a conventional associative memory are arranged in the form of an array, there is no way to tell whether one word is next to the other or not. This is because each word is independent and isolated from other words. This independence and isolation poses two limitations to a conventional associative memory. A conventional associative memory cannot be used to search a multi-word string that requires storage space of more than one word. Similarly, it cannot be used to retrieve a multi-word string. The example shown in (Fig. 1) illustrates these two limitations. Although the string "abcdefghi" is stored in word 2 and word 3, there is no way to search for that string. The reasons are that the string requires storage space of more than one word, and there are no mechanism in the conventional associative memory to tell whether one word is next to the other or not. The same reasons apply to the retrieval of multi-word string.

A conventional associative memory can be use as a table look-up device, but these two limitations make the table look-up device inflexible. A keyword "abcd_", for example, can be searched (where the bits represented by '_' are not used for comparison) (see Fig. 1). Word 1 matches the keyword. It is retrieved, and p2 can be used as an address to a record stored elsewhere. Since only one keyword can be searched, the number of bits in one word (m in this example) determines the maximum length of both the keyword and the pointer. However, sometimes a long keyword and a long pointer is required, sometimes a relatively short one is

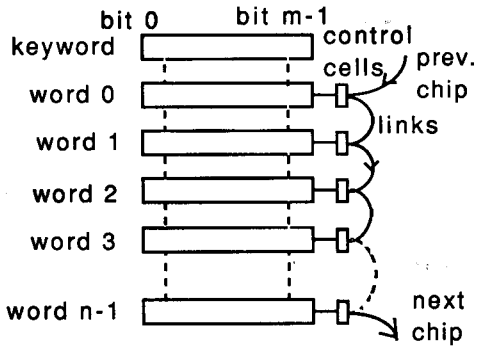

Fig. 2. Link-words association in GAM.

sufficient. For general purpose applications, one cannot decide what the maximum length of a word should be. If the bit length is too long, only a few words can be implemented in an associative memory. If it is too short, the associative memory may not meet the requirement of many applications.

In order to make an associative memory suitable for general purpose applications, several designs have been proposed, attempting to make associative memory perform the multi-word string search and retrieval. However, these designs pose their own limitations. One design limits the string length to eight words [8]. Another design requires all the strings stored in the memory to have same length. Also, the address of the first word of each string needs to be known before the search [9].

The design of a general purpose associative memory called GAM (Generic Associative memory) is discussed in this paper. The design is motivated by recognizing the need to handle in a flexible manner multi-word string comparison and to retrieve multi-word data associated with the matched strings. GAM implements both functions with a unique link mechanism. The control cells (see Fig. 2) not only serve as tag bits but link each word to its next word. With this link mechanism, the associative memory can search a string or a record with any number of words, and can then retrieve from the memory any number of pointers or data associated with the matched records (limited by the total memory space in all the connected memory chips). GAM can also be used as a standard RAM for random read/write operations, and as a conventional associative memory for relational search and pattern matching.

In Section II, we first outline two applications of this specially design associative memory, namely, Prolog clause indexing and Relational database search-retrieval operations. In Section III, we define the input and output lines of a GAM chip, followed by a discussion on the modes of operations in Section IV. In Section V, we describe the circuit design and implementation, while the evaluation is given in Section VI. Finally, we give a conclusion of our work and suggest future research.

II. Applications of GAM

The ability of GAM to perform parallel search/retrieval of multi-word string makes GAM an excellent device for Prolog clause indexing and for Relational database search/retrieval operations. These two applications of GAM are outlined as follows.

A. Prolog Clause Indexing

A Prolog program consists of a finite set of clauses which are facts, rules, and queries [10]. In the execution of a Prolog program, the unification engine needs to find the right clauses to unify. This is usually done by compiling the Prolog program and by using a hash table. For example, in Warren Abstract Machine's instructions set, *try*, *retry*, and *trust* instructions, followed by clause addresses, and *switch_on* instructions, followed by pointers to a hash table, are used to index the clauses [11]. The compiled program usually lacks the flexibility for modification. Every time one modifies the program, the program has to be recompiled. An interpreter is usually more suitable for dynamic environment, such as during the development of a Prolog Program. With the help of GAM, a unification engine can directly interpret the source program with reasonable speed.

There are two ways we can use GAM for Prolog clause indexing. If a Prolog program can be completely stored in GAM chips, GAM can directly search all the clauses and retrieve the matched ones. Consider the example shown in Program 1. A Prolog program is stored in GAM,

```
.father(peter,john).male(john).father(peter,marry).female(marry)
.son(X,Y):-father(Y,X),male(X).daughter(X,Y):-father(Y,X),female(X).
```

Program 1. Complete program search.

one clause after another as shown. Each character is stored in one eight-bit word in GAM. The clause boundaries are designated with a special character. In the examples, we use a period for convenience. Given a query, son(Who,peter), GAM searches the keyword .son(and then retrieves its arguments and its clause body, X,Y):-father(Y,X),male(X). Similarly, given a query, daughter(marry,peter), GAM searches the keyword .daughter(and then retrieve, X,Y):-father(Y,X),female(X). A unification engine can then use the retrieved data to finish the unification process.

In case the whole Prolog program cannot be stored in GAM. The name of each clause head followed by a pointer will be stored in GAM, while the arguments and the clause bodies are stored in standard RAM (see Program 2). GAM can perform parallel search on the names of the clause head and then retrieve the corresponding pointers which are the addresses of the arguments and the clause bodies. A unification engine can then use the retrieved pointers to access the RAM and to complete the unification process. A detailed description of how GAM can perform these parallel search and retrieval is given in Section IV.

B. Relational Database Search/Retrieval

For relational database applications, GAM can perform parallel search of a key attribute and then retrieve the next attribute in a way similar to that for Prolog clause indexing. Furthermore, it can be programed to do relational searches, such as finding the records which are greater than, greater than or equal to, or less than, some specified value. Several algorithms for doing such relational searches were described in detail in [12]. Although those algorithms were designed for single keyword search, they could be extended to do multi-word string search for GAM.

IN GAM	IN RAM
.father(p1	p1: peter,john).
.male(p2	p2: john).
.father(p3	p3: peter,marry). .female(p4
	p4: marry).
.son(p5	p5: *X,Y) :- father(Y,X), male(X).*
.daughter(p6	p6: X,Y):- father(Y,X), female(X).

Program 2. Program in GAM and RAM.

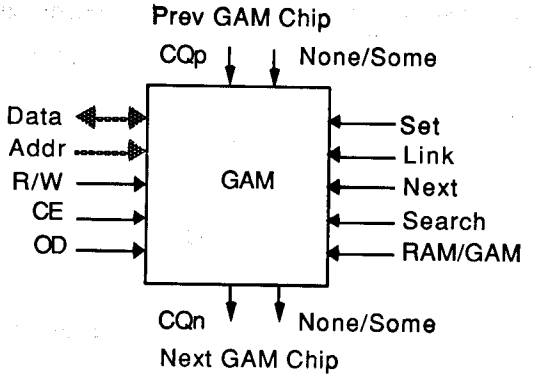

Fig. 3. GAM Chip (Inputs & Outputs)

III. Input/Output Lines of GAM Chip

Beginning from this Section, the design and operations of GAM are discussed. The input and output lines of a GAM chip are defined in this Section.

Fig. 3 shows the input and output lines of a GAM chip. Data, R/W, Addr, CE, and OD are the input and output lines used in a standard RAM. Set, Link, Next, Search, and RAM/GAM are the control lines used for associative search. CQp and None/Some are the lines which are used to concatenate several GAM chips. None/Some is also the signal line which tells whether there are no matched word or some matched words.

The control lines are defined as follows: Set turns on all control bits to prepare GAM for associative search. Link prepares GAM to search the next word during a multi-word string search, and it sets up a read-pointer, which is realized by a control bit, to point to the next word during a multi-word string retrieval. Next updates the read-pointer to point to the next string that also matches the key string. If more than one string matches the key string, this command allows all the data or pointers corresponding to those strings to be retrieved. Search enables GAM for parallel search and pattern matching. RAM/GAM selects GAM chip to act as a standard RAM or to act as associative memory.

```
                                ->time
     . A b c (                  <-data
    SMLMLMLMLMLRNRN             <-commands
. 1100000000000000
m 1010000000000000
n 1000000000000000
( 1000000000000000
p 1-0000000000000000
. 11-000000000000
A 1011000000000000
b 1000110000000000
c 1000001100000000
( 1000000011000000   first
p 1-0000000011000000 <-read
. 11-000000000000
x 1010000000000000
y 1000000000000000
( 1000000000000000
p 1-0000000000000000
. 11-000000000000
A 1011000000000000
b 1000110000000000
c 1000001100000000
( 1000000011000000    next
p 1-0000000011111000 <-read

^-memory words ^-control bits
```

Fig. 4. GAM operation example. (p is pointer, - is unknow 1 or 0)
(commands: Set, Match, Link, Read, Next)

IV. Modes of Operations

GAM can perform three major modes of operations, namely, standard RAM mode, Single-Word Associative mode, and Link-Words Associative mode.

In standard RAM mode, GAM functions the same way as RAM. This mode provides GAM simple interfacing to disk drive controller, DMA channels, and conventional processors (eg., for data loading and unloading).

In Single-Word Associative mode, GAM functions in the same way as a conventional associative memory. In this mode, the search procedure, in its simplest case, is a three-step process. First, the keyword is loaded into the Comparand register. Secondly, a Search command is given. Finally, the None/Some output line will tell whether there are some words in the GAM matched the keyword.

The Single-Word Associative mode is described in more detail as follows. First, the keyword is loaded into the m-bits Comparand register. Each bit of the keyword can be masked ON or OFF by the corresponding bit in the Mask register (the keyword shown in Fig. 2 consists of Comparand and Mask registers). Only the ON bits of the keyword will be compared. All the

words in the GAM are compared with the Comparand register concurrently. The results of the comparison are stored in the Control Cells (see Fig. 2). The matched words can then be read out or written over, one at a time, using the Next command. The None/Some output line signals either "None" matched word or "Some" matched words.

In Link-Words Associative mode, GAM can perform parallel search/retrieval of multi-word strings. In this mode, the search procedure, in its simplest case, is outlined as follows: The key string is entered, one character at a time, into GAM. A Search command is issued after each character. A Link command is given after each Search command, except for the last one. After all the characters of the key string have been entered, the None/Some line will tell whether there are some strings in the GAM which match the key string.

For example, consider the Prolog program shown in Program 2. Given a query, father(Who,marry), GAM is used to search the string ".father(", and to retrieve both the pointers p1 and p3. The first character '.' is loaded into the Comparand register. A Search command is given. Then, a Link command is given. The second character 'f' is next loaded into the register. A Search command followed by a Link command is given. This process continues until the last character is encountered. The last character '(' is again loaded into the register and a Search command issued. At this point, the None/Some output line will tell there are some strings in the GAM that matched the string ".father(". A Link command will set a Read-pointer pointing to p1. Then, a Read command is given. As a result, the pointer p1 is retrieved. A Next command is then issued, which sets the Read-pointer pointed to p3. Finally, a Read command will retrieve the pointer p3.

The detailed control signals of the Link-Words Associative mode are described in the following example. Fig. 4 shows an example for searching a string, .Abc(, and retrieving the pointers, p's. GAM contains the words shown in the first column of the figure. The Mask register is loaded with all 1's. The first row in the figure shows the data stored in the Comparand register in the corresponding steps. The second row in the figure shows the commands in every step of the operations.

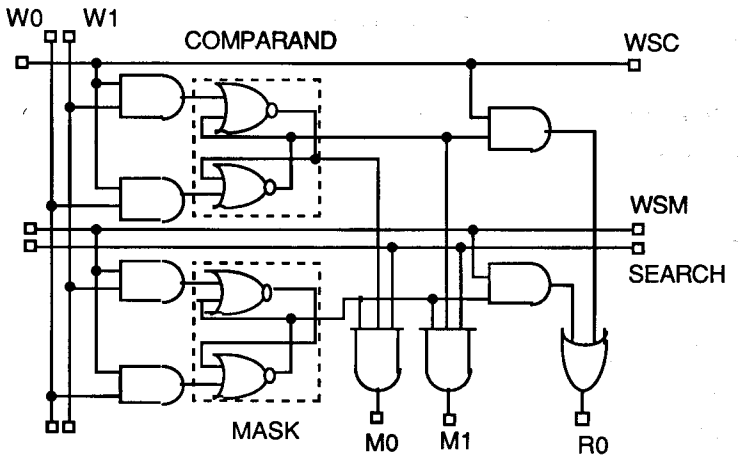

Fig. 5. Comparand and mask cell.

The first command to the GAM chip is Set, which sets all the Control Cells to 1's (second column in Fig. 4). The character '.' is loaded into the Comparand register. A Search (Match) command is then given. All the Control bits corresponding to the mismatches will be reset to 0's (third column). A Link command is then given. As the result all Control bits are shifted one word position down, with the first Control bits set to 0. The character 'A' is then loaded into the Comparand register. A Search (Match) command followed by a Link command are then given. The cycle repeats till the last character.

Then, a Read command is given and the data pointed by the First-Read (see Fig. 4) will be read out. The command Next is then given. As a result the Control bit corresponding to the First-Read position will be set to 0. And, the position of the Read-pointer will be updated. Now, the Read-pointer points to the Next-Read position. The None/Some output line will signal that there is still another match. A second Read command is then given, and the data pointed to by the Next-Read will be read out. A second Next command is given. After that, the None/Some line will signal no more matches.

V. Circuit Design and Implementation

The gate-level circuit design to accomplish the modes of operations is described in this Section.

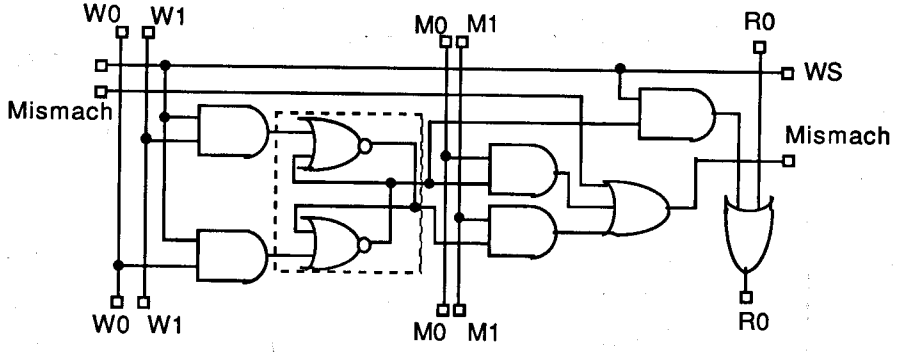

Fig. 6. Associative cell.

Each bit in the keyword (see Fig. 2) consists of a Comparand and Mask cells. The circuit to implement the Comparand and Mask cells is shown in Fig. 5. This cell can be read or written using the same method as used in standard RAM. They provide the masked comparand bit M0 and M1 to all the associative memory cells in the corresponding column.

Each bit in word 0 to word n-1 (see Fig. 2) is realized by an Associative cell. The Associative cell shown in Fig. 6 is a basic associative memory cell [13]. With a few minor modification on the control circuit, the associative cell can be implemented with 9 transistors on VLSI or LSI as shown in Fig. 7 [8; 9; 14; 15].

The GAM Control Cell (see Fig. 8) [16] which realizes all three modes of operations transpires to be amazingly simple. Only 10 gates are needed for each control cell. All the control signals to the Control Cell, namely, Link, Set, Next, and RAM/GAM, are positive pulses and no two control pulses can be given at the same time.

To realize the RAM/GAM selection, the gate O1 with the control line RAM/GAM is used. While all other control lines are pulled low, a positive pulse to RAM/GAM control line will reset all the control bits to 0's. The word select lines can then be used to select the word to be read or written in the same way as in a standard RAM. While the RAM/GAM is pulled low, Set control line can be used to set all the control bits to 1's which will make the GAM ready for associative search.

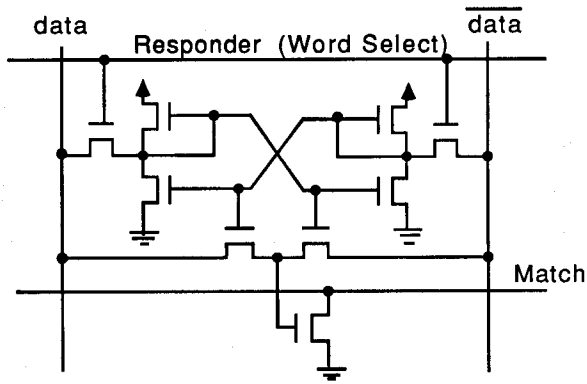

Fig. 7. A 9-transistor associative cell.

To realize the Link command, the gates A1, A2, the memory cell, and the Link control line act like a shift register. All the control bits are shifted down one position in the Link operation. The bit in the previous control cell is shifted down to the present control cell while the bit in the present control cell is shifted down to the next one.

To realize the None/Some signal line and the Read-pointer, gates A3, A4, which are connected to the previous and the next control cells, act like a priority encoder. The Read-pointer is pointed to the highest priority word which is the word to be read out or written over. To realize the Next command, the gate A5 and the Next control line will reset the memory cell which is selected by the Read-pointer. As a result the Read-pointer will point to the next matched word.

VI. Evaluation

The circuit design of GAM has been verified by using a simulation program called LogicWork run on a Mac II. In string search operations, the speed up of this associative memory over a conventional computer with RAM is proportional to the number of words stored in the memory. For example, suppose that there are 100 words stored in the memory, and the search key is 5 words. Also, assume that the same search algorithm is used in both systems. The associative memory described here requires 10 steps while the conventional computer with RAM requires more than 100 steps. If the number of words stored in the memory is now increased to 500 words. GAM still requires 10 steps. But, the conventional computer now requires more than 500 steps. In this example, if the access time of GAM is 1.5 times that of RAM, the overall speed up of GAM over RAM is over 33 times.

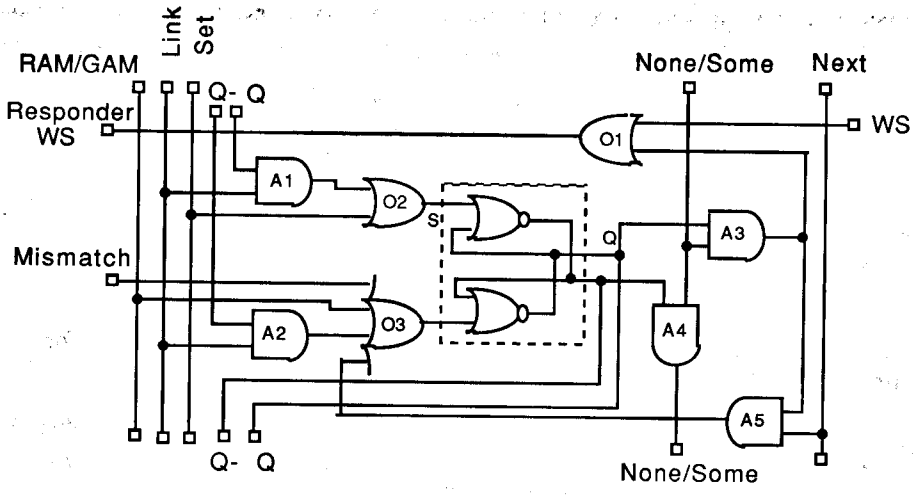

Fig. 8. GAM control cell

VII. Conclusions and Future Work

The design and applications of an associative memory called GAM, which can perform parallel search/retrieval of multi-word string, were described. GAM can also function as a standard RAM and as a conventional associative memory. These features make GAM suitable for general purpose applications, as well as an excellent device for Prolog clause indexing and Relational database search/retrieval operations. We are incorporating GAM to our Prolog Knowledge-Base Computer (PKBC) which is being designed as a symbolic multi-processor system with hardware support for unification. Our future research includes the design of a unification engine for performing full unification. The unification engine is based on a pattern matching device developed for complex pattern matching [17]. We need to extend it to handle such things as variables and structures and to incorporate it with GAM which can best serve as a clause indexing device.

References

[1] Char, Jois Malathi, Vladimir Cherkassky, Harry Wechsler, and George Lee Zimmerman, "Distributed and Fault-Tolerant Computation for Retrieval Tasks Using Distributed Associative Memories," IEEE Transactions on Computers, Vol. 37, No. 4, Pages 484-490, April, 1988.

[2] Lee, Dik Lun, "A Distributed Multiple-Response Resolver for Value-ordered Retrieval," Proc. 12th Annual International Symposium on Computer Architecture, Pages 258-265, June, 1985.

[3] Robinson, Ian, "A Prolog Processor Based on a Pattern Matching Memory Device," Proc. International Conference on Logic Programming, Pages 172-179, 1986.

[4] Anderson, Judy M., William S. Coates, Alan L. Davis, Robert W. Hon, Ian N. Robinson, Shane V. Robison, and Kenneth S. Stevens, "The Architecture of FAIM-1," Computer, Pages 55-65, January, 1987.

[5] Moldovan, Dan I. and Yu-Wen Tung, "SNAP: A VLSI Architecture for Artificial Intelligence Processing," Journal of Parallel and Distributed Computing 2, 109-131, 1985.

[6] Naganuma, Jiro, Takeshi Ogura, Shin-Ichiro Yamada, and Takashi Kimura, "High-Speed CAM-Based Architecture for a Prolog Machine (ASCA)," IEEE Transactions on Computers, Vol. 37, No. 11, pages 1375-1383, November 1988.

[7] Lee, Dik Lun, and Lochouskey, Fred, "Text Retrieval Machines," in Office Automation, D. Tsichritzis, ed., Spring Verley, New York, N.Y., pages 339-375, 1985.

[8] Ogura, Takeshi, Shin-ichiro Yamada, and Junzo Yamada, "A 20Kb CMOS Associative Memory LSI for Artificial Intelligence Machines," IEEE International Conference on Computer Design, Page 574-577, 1986.

[9] Adams, Stuart J., Mary Jane Irwin, and Robert M. Owens, "A Parallel General Purpose CAM Architecture," Advance Research in VLSI, Pages 51-71, 1986.

[10] Sterling, Leon and Ehud Shapiro, The Art of Prolog. The MIT Press, Cambridge, Massachusetts, 1986.

[11] Warren, David H.D., "An Abstract Prolog Instruction Set," Technical Note 309, SRI International, October, 1983.

[12] Foster, Caxton C., Content Addressable Parallel Processors. Van Nostrand Reinhold Co., New York, pages 57-102, 1976.

[13] Foster, Caxton C., Content Addressable Parallel Processors. Van Nostrand Reinhold Co., New York, page 15, 1976.

[14] Robinson, Phillip, "The SUM: An AI Coprocessor," Byte, Pages 169-180, June, 1985.

[15] Kokubu, Akio, Minoru Kuroda and Tatsumi Furuya, "Orthogenal Memory A step Toward Realization of Large Capacity Associative Memory," VLSI 85, E.Horbst (editor) Elsevier Science Publisher B.V. (North-Holland), 1986.

[16] Choi, C.H. Ben, "Parallel Distributed Computer Architecture and General Associative Memory for Artificial Intelligent Processing," Master thesis, Dept. of Electrical Engineering, The Ohio State University, Columbus Ohio, 1988.

[17] Lee, D.L. ALTEP - A cellular processor for high-speed pattern matching. New Generation Computing, Vol. 4, No. 3, pages 225-244, Sept., 1986.