VLSI Design
1994, Vol. 2, No. 3, pp. 267–286
Reprints available directly from the publisher
Photocopying permitted by license only

Modular Scheme for Designing Special Purpose Associative Memories and Beyond

A.R. HURSON and S. PAKZAD
Computer Science Engineering Department, The Pennsylvania State University, University Park, Pennsylvania 16802

(Received April 20, 1991, Revised March 15, 1993)

The use of associative memories—storage devices that allow data retrieval based on contents—has often been suggested to speed up the performance of many applications. Until recently, using such **content-addressable** memories (CAMs) was unfeasible due to their high hardware cost. However, the advent of VLSI has made the class of fully-parallel associative memory cost-effective for implementation. This paper briefly overviews design of several fully parallel associative memories proposed in the literature, concentrating on the design of fully-parallel θ-search CAMs.

Existing market realities require that product development be fast and predictable. As a result, design flexibility and automation are becoming increasingly important design features. Using the various CAM designs reviewed, the paper collects the features of these designs into a general, modular CAM organization and describes its major components. The modular CAM organization can be used to design application specific CAMs of varying degrees of functionality. Design and space complexity of a sample associative memory suitable for relational database operations is studied. Finally, the application of genetic algorithms as a means to developing automated design tools for fabrication of modular VLSI design chips is discussed.

Given a library of CAM modules, the desired functionality and a set of speed and area constraints, this optimization technique produces a suitable CAM design. The proposed technique has been implemented and its performance measure is briefly addressed.

Key Words: *Content addressable memory, Associative memory, θ-Search cell, VLSI fabrication, Automated design tool, Genetic algorithms*

I. INTRODUCTION

Since mid 1960s computer architects have experimented the prospect of using smart memories in their systems. In such memory organization, the storage capability is enhanced with logic to allow direct execution of basic functions directly on stored values. This virtually eliminated **transportation** of data from memory storage to an arithmetic/logic unit—a persistent bottleneck in conventional systems.

One specific type of smart memory, a **content addressable memory** (CAM) allows access to the contents of the memory based on data values rather than the addressees. Essentially, a standard random access memory device is augmented to allow search operations to be performed over the data stored in the memory. Yau and Fung [23] have discussed about four classes of CAMs: **fully parallel, bit serial, word serial,** and **block oriented.** Such a classification is based on the amount of parallelism incorporated in the search operation and thus reflects the tradeoff between the size (cost) and speed of the associative memory.

The fully parallel CAM has search circuitry associated with every bit in the memory. This lets the entire memory be searched at the same time and provides the fastest search time of all the classifications. The bit serial approach has search circuitry associated with a single bit of each word and all the bits of each word must be shifted through its search bit to perform a search. The word serial CAM has search circuitry associated with a single word of the memory, thereby implementing a hardware version of a standard linear search algorithm. Finally, the block oriented approach has search circuitry associated with a block of data at the secondary storage level. Usually this is implemented by adding a processor to the read/write head of a disk; this processor

can perform associative operations on the data passing under the head.

In the past, the additional cost and hardware complexity of associative memories have been the major factor to limit the size of such systems and as a result, have prevented their widespread use by designers. However, with the advent of VLSI technology, such organizations can now be cost-effective and should be re-examined. In particular, the fully parallel associative memory organization due to its modularity, simplicity, regularity, and speed seems a promising candidate for VLSI implementation.

This paper first emphasizes the need for general purpose associative memories by overviewing several applications where CAMs can significantly affect the performance. The design of a high-performance/high-capacity θ-search associative cell is proposed and analyzed. The paper then presents motivations for developing a general, modular CAM organization suitable for fabrication of high-capacity associative memories with varying degrees of functionality. Finally, the application of the genetic algorithms as an automation tool in the design and fabrication of VLSI chips in general is motivated and its application in the design of fully parallel associative memories as a test-bed will be analyzed.

II. ASSOCIATIVE MEMORY APPLICATIONS

Ever since associative memories were proposed over 30 years ago, many designers in diverse application areas have suggested the use of a CAM's parallel processing and its content accessibility to improve their performance. This section briefly overviews a few applications which utilize CAMs in their designs to increase performance.

The use of CAMs to improve the performance of memory management is already well established [4]. Associative memories can be used to quickly execute the table entry look-up and modification operations used in memory management systems. For this reason, CAMs are often used as translation look-aside buffers in virtual memory systems and as tag directories in fully-associative cache organizations. For both these applications the CAM needs to perform equality searches on its contents.

Associative memories have often been used in the architecture of database machines [8]. The parallel search capabilities of CAMs make these devices ideally suited for the database environments [2, 7, 17]. Typically, a CAM used for database operations should have at least maskable equality-search, maskable write, and multiple write capabilities. However, many database applications often perform θ-searches (where θ is the element of the set $\{<, >, \leq, \geq, =, \neq\}$). As a result, making an associative memory which can implement a θ-search directly in hardware is very desirable.

Associative memories are also being used in the design of the prolog machines for efficient handling of backtracking and unification operations [14, 15, 20]. Nahanuma et al., [14] show how a CAM with a maskable equality search, maskable write, and the garbage collection abilities discussed in [15] can reduce the backtracking time to a small, constant value regardless of the number of bindings. Stormon et al., [20] also show how a CAM can be used to speed performance on unification through clause filtering. Finally, we have recently witnessed a surge of interest in the application of associative memory and associative processing in the area of Computer Vision [5].

III. EARLIER FULLY PARALLEL CAM DESIGNS

VLSI technology has meant that enough devices can finally be packed onto a single chip to make a fully parallel associative memory feasible. Such a memory is very well suited to VLSI implementation because of its simple, regular, and modular structure. Because of this, several fully parallel CAM designs suitable for current technologies have been proposed in the literature [6, 10, 11, 15, 16, 18, 20]. Physical characteristics and intended application areas of some of these designs have been summarized in Table I. For further information interested reader is referred to [3].

Recognizing the need for fast θ-searches in the area of database processing, [6] proposed a fully parallel maskable θ-search CAM suitable for VLSI technology. The CAM uses search circuitry at each bit position to determine the two θ-relations, less-than and greater-than, for each word in a cascade fashion from the most significant bit to the least significant bit (Figure 1). All of the other θ-relations can be determined from these two values. The design also includes a multiple maskable write capability. However, it did not address in detail the problem of preventing empty associative words from participating in a search. The CAM uses nMOS technology and its associative search cell size was estimated at $5000\lambda^2$.

The CAM design presented in [18], like the CAM of [6], is a fully parallel maskable θ-search CAM

TABLE I
Characteristics of Some Fully Parallel Associative Memories

Year	Capacity (bits)	Cell Size λ^2	Application
1985[15]	4 K	2652	Artificial Intelligence
1985[11]	8 K	1080	Dataflow Processing
1986[16]	20 K	1438	Artificial Intelligence
1987[6]	16 K	5000	Database Processing
1988[18]	6 K	14092	Database Processing
1988[20]	2 K	3040	Prolog Machine
1988[10]	9 K	1656	Parallel Processing

intended for database applications. However, unlike in [6] where the hardware search time is O(m) where m is the width of the associative memory in bits, this new design has a search time of O(A) where A is the length of the attribute being searched which can be much smaller than m. This is possible due to using domino CMOS and pull down logic attached to precharged less-than, greater-than, and equality busses for the search circuitry instead of static CMOS logic (Figure 2). The associative cell size (minus the memory element) can be estimated at $7280\lambda^2$ when routing is taken into consideration.

IV. AN IMPROVED Θ-SEARCH ASSOCIATIVE CELL

A high performance θ-search CAM, with its increased functionality over the standard equality-search CAM, is conceptually a good idea for CAM development. However, the θ-search bit cell proposed in [18] contains over 7.5 times the number of transistors found in a static RAM cell. Despite its increased functionality, this may be considered too high a price to pay for the associative memory. Since a θ-search design is advantageous in some applications, we tried to redesign this bit cell to reduce its size to a minimum.

The result is a θ-search cell evolved from the [18] design. Like the original design, our cell utilizes precharged busses to make the search time proportional to the size of the attribute field being searched, i.e., O(A) where A is the number of bits in the attribute field. This design allows any one set of contiguous bits to be searched over at a time. However, the new cell requires only 15 transistors per bit for the search circuitry instead of more than 40 transistors per bit found in the [18] design.

The logic used in this design has one signal propagating from cell to cell. Specifically, an **equality-so-far** (EQ) signal is generated in each cell and is passed onto its less-significant neighbor via domino CMOS logic. Associated with each bit position i are four signals: c_i, $\overline{c_i}$, $\overline{mask_i}$, and FI_i. The c lines are generated by the equations $c_i = comp_i \cdot mask_i$ and $\overline{c_i} = \overline{comp_i} \cdot mask_i$. FI_i is a finish signal indicating the last position of the search attribute and is generated by $mask_{i-1} \cdot \overline{mask_i}$. In addition, three precharged busses ($\overline{GT}$, $\overline{LT}$, and $\overline{EQL}$) are associated with each word j and can only be pulled down as follows:

$$\overline{LT} \text{ pulled low when } c_i \cdot \overline{b_{i,j}} \cdot EQ_{i-1,j} = 1$$

$$\overline{GT} \text{ pulled low when } \overline{c_i} \cdot b_{i,j} \cdot EQ_{i-1,j} = 1$$

$$\overline{EQL} \text{ pulled low when } FI_i \cdot EQ_{i-1,j} = 1$$

where $b_{i,j}$ and $\overline{b_{i,j}}$ are the true and complemented values of bit i of word j. Figures 3 and 4 show the circuitry for each bit in this design. The special Least-Significant-Bit (LSB) cell is required because of the way the finish signal is generated.

A fast search time is achieved by using the $\overline{mask_i}$ and the FI_i signals. The $\overline{mask_i}$ signals cause all of the bit positions not participating in the search to simultaneously pass active EQ signals to their least significant neighbor's search circuitry. However,

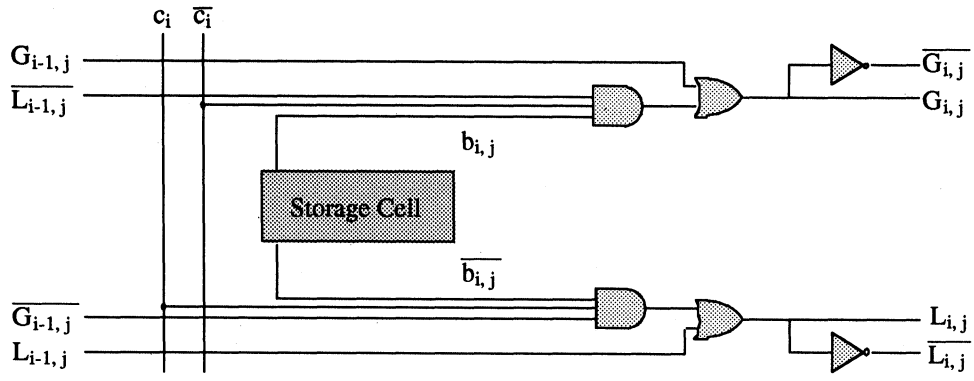

FIGURE 1 Theta-search static logic design.

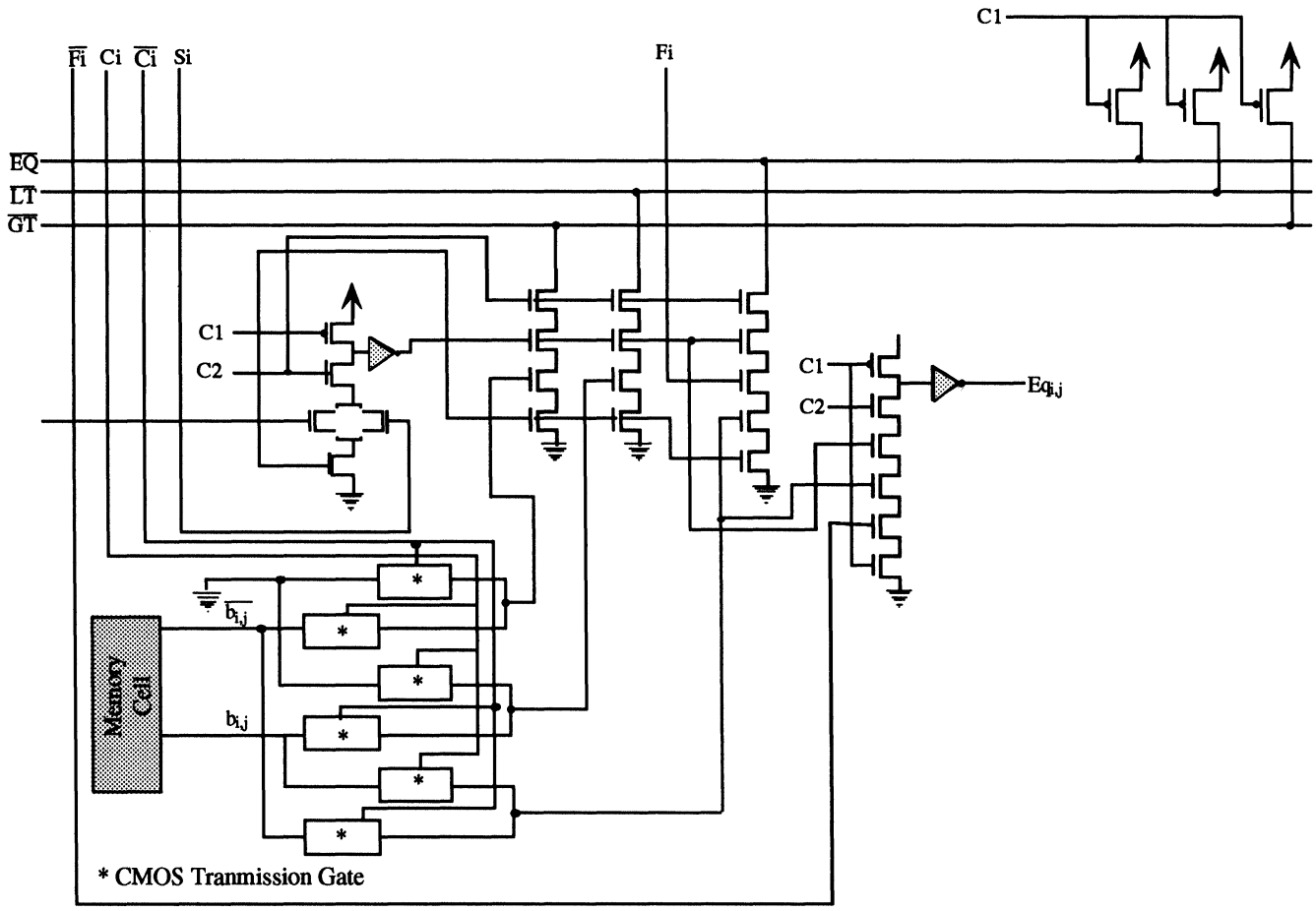

FIGURE 2 Theta-search domino CMOS design.

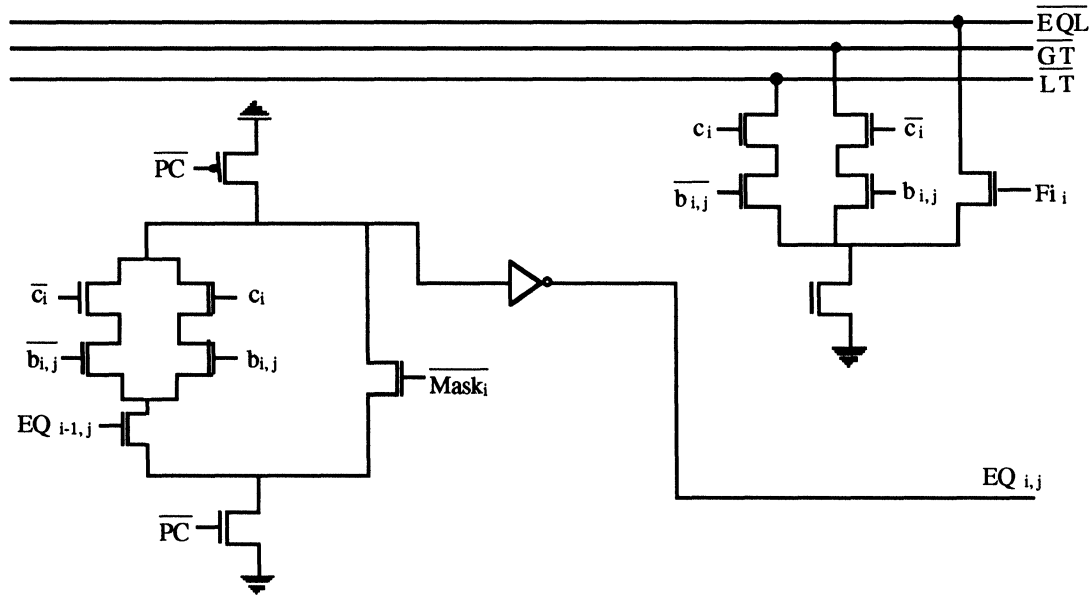

FIGURE 3 CMOS with precharge bus design (all but LSB).

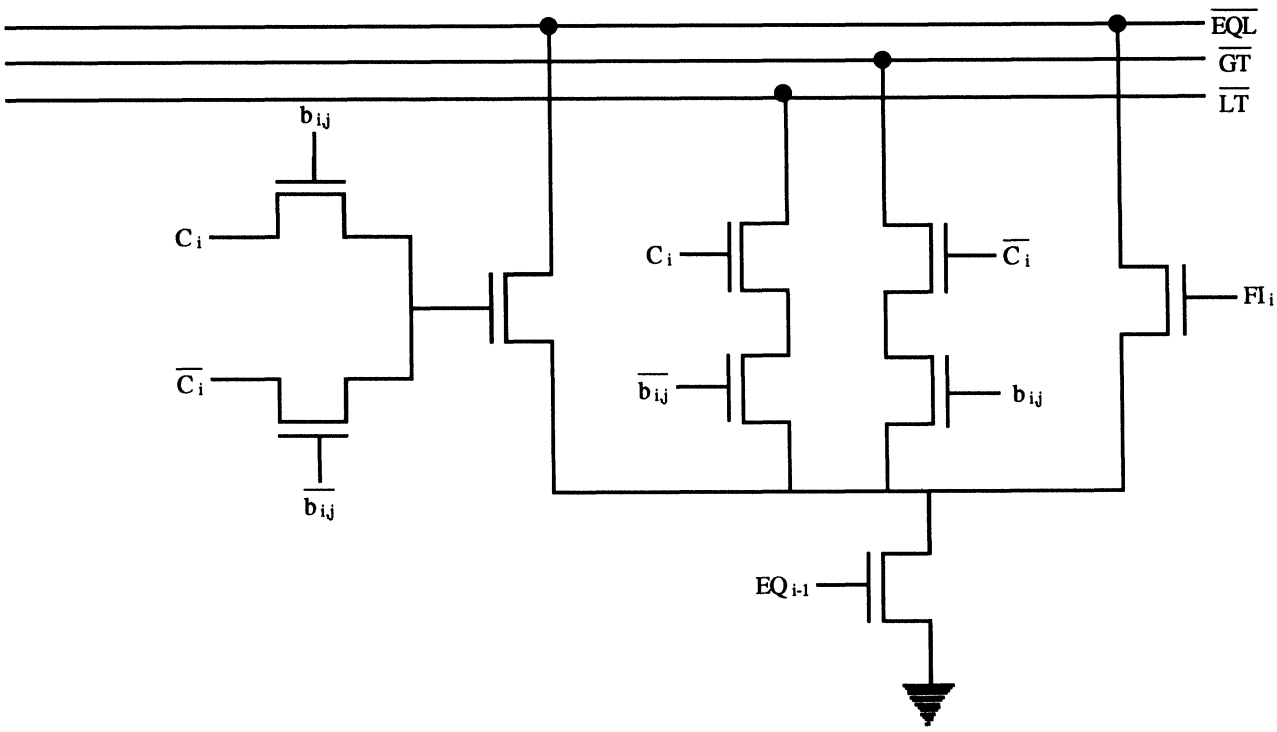

FIGURE 4 CMOS with precharge bus design (LSB).

since the bits not participating in the search (i.e., bit i where $\text{mask}_i = 0$) have both their c lines (c_i or $\bar{c}_i$) low, their search circuits are unable to pull any of the precharged busses low. Thus the EQ signals effectively activate only the search circuitry of the first bit of the attribute to be searched. The EQ signal then propagates through the neighboring bit modules of the search attribute (most significant bit to least significant bit) until one bit of the attribute indicates inequality. When this happens, the appropriate $\overline{GT}$ or $\overline{LT}$ bus will be pulled low. If, on the other hand, the word's attribute is equal to the comparand's attribute, then the EQ signal output from the last bit of the attribute will be high. Since an active FI signal is sent to the bit immediately following the last bit of the search attribute, this causes the $\overline{EQL}$ bus to be discharged (indicating equality). Thus at the end of the search one of the three search busses will be pulled low to indicate the result of the search for that word.

Since the transistors in the search circuitry are used only to discharge precharged nodes (either a bus or the gate of a CMOS inverter), the transistors can be minimum sized (e.g., 2λ by 2λ) to reduce space without seriously affecting performance. If we use the 8-transistor maskable, multiple write memory cell described in section VII.2 (Figure 9), then the total active area per bit would be $92\lambda^2$. To compensate for layout inefficiencies due to routing problems, this first order area estimate is multiplied by a factor of 60. Thus the estimated size of this proposed associative cell is $5520\lambda^2$ with the memory element ($3600\lambda^2$ without it). This is much smaller than the [18] design (which is $7280\lambda^2$ without the memory element) and is just 10% larger than the slower design of [6]. Note that this design in also only 3.5–5 times larger than the equality-search CAMs described in [15], a reasonable cost considering the functionality and performance provided by the new design. Table II compares and contrasts the physical characteristics of different θ-search associative memories.

V. GENERAL CAM ORGANIZATION—MOTIVATION

As discussed in the previous sections, CAMs can be used in an architecture to significantly speed up the

TABLE II
Physical Characteristics of Θ-Search Associative Memories

Model	Size Including Memory Element (λ^2)	θ-Search Times (ns)	Capacity+ (K bits)
Ref. 6	5000	139	16
Ref. 18*	14092	67	5.7
Proposed* Scheme	9544	40	8.4

Word length = 64 bits
Attribute length = 16 bits*
Chip Memory Area = 11.6 × 7.3 mms^{2+}

processing rate of a computer systems for a particular application. In addition, recent advances in technology has made it economically feasible to design and fabricate high capacity/high performance associative chips. Yet despite these two facts, there are very few associative memories currently available in the market either as general purpose chips or as components in standard cell libraries for VLSI design. The perceived high cost of associative memory could have contributed to this fact. In some cases, though, this size and cost increase should be acceptable. For example, the equality-search bit cell proposed in [15] contains only about twice as many transistors as a standard CMOS static RAM cell, a penalty more than offset by the increased functionality of the associative memory. Nevertheless, such a belief might not be the only reason that has discouraged the mass production of associative memories. As has been noted, applications which use associative operations often require associative memories with different functionality. Therefore, it is necessary to develop a scheme which allows easy and fast implementation and fabrication of associative memories which can support the needed operations efficiently—i.e., a design methodology which offers a high degree of modularity, independence and compatibility among different elements of an associative memory.

A modular design is a way of dividing a system into functional components that interface with each other in the most efficient manner. Components that require a significant amount of communication are best implemented in the same module. Each component has to receive well-specified inputs, perform a certain well-defined function, and output its results to other modules in a useful and recognizable form.

The advantages to this approach are manifold. Because of the well defined interfacing between modules, replacement of problematic modules is simplified greatly. Flexibility is enhanced since new modules can be plugged in to provide corrected functionality or better performance characteristics. Work can be effectively divided into independent subtasks that can be tackled in parallel, without much interference. Testing procedures also benefit from this organization since problems can be isolated to single modules and their internal workings.

An added advantage to modular design emerges from the collection of system modules that have already been implemented in the past. These can readily be integrated into a new modular design that requires a similar type of functionality, without the expenditure of rebuilding the module from scratch. Thus, modularity allows projects to share designs and even parts from related projects, producing the equivalent environment in hardware as object-oriented programming does in software.

To help simplify the development of different types of CAMs, we believe that a general and modular CAM organization suitable for creating CAMs of various degrees of functionality is needed. Ideally, this organization should contain a high-level CAM architecture composed of a set of mostly-independent modules and a list of common features shared by every CAM regardless of its functionality. These common features are implemented by modules whose designs usually remain the same regardless of the type of CAM being developed. A set of special-purpose features, different for each CAM implementation, determines the exact functionality of the CAM. These features are implemented by special-purpose modules in the CAM.

A general, modular CAM organization has two benefits over specialized CAM designs. The first is that such an organization would guarantee that each completed CAM design, regardless of its functionality, would contain common features needed by all types of associative memories. This is important because some of the specialized CAM designs do not address some important issues associated with CAMs. For example, both [6] and [18] do not really discuss how to prevent empty associative words from participating in a search, a common feature required by all CAMs. The second benefit of a general CAM organization is that CAMs of varying degrees of functionality can be easily designed by merely replacing certain special-purpose modules in the CAM with new modules which implement the needed functions.

VI. A GENERAL CAM ARCHITECTURE

The previous section discussed the motivation behind developing a general, modular CAM organization. This section presents such an organization by closely examining the CAM designs proposed in the literature and incorporating the common as well as the most useful features, as implied by the applications. Figure 5 shows the organization of a modular CAM composed of the following modules: Comparand register, mask register, associative word array, decoder, I/O interface, tag flags, word select flags, empty flags, a multiple match resolver, and enable inputs and outputs:

The comparand and mask registers hold values used during an associative search. The comparand register holds the value to search the associative

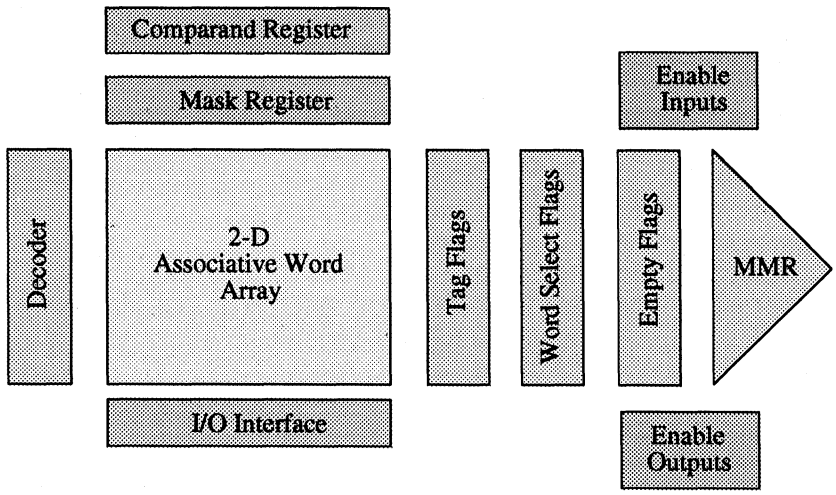

FIGURE 5 Proposed general CAM organization.

words against, while the mask register holds a bit pattern determining which bits of the associative words participate in the search:

$\forall\ 1 \leq j \leq$ no. of bits in a word

$m_j = 1$ bit j participates in search,

$m_j = 0$ bit j does not participate in search.

If a maskable write capability is allowed, the bit pattern in the mask register also determines what fields of an associative word are modified.

Data is held in a 2-D **associative word array** consisting of a linear array of n m-bit words. Each bit cell contains a single-bit storage element and some search circuitry. Since for large memories this array comprises most of the circuitry of the CAM, the size of the bit cell should be as small as possible to reduce the overall cost of a CAM.

Attached to each associative word are three one-bit flags: an **empty flag** (EF), a **tag flag** (TF), and a **word select flag** (WS). The contents of the empty flag reflect whether or not the associative word is empty:

$\forall\ 0 \leq i \leq n - 1$

$EF_i = 1$ word i is empty,

$EF_i = 0$ word i contains valid data.

Empty associative words do not participate in associative searches. An empty flag is cleared by writing to its designated associative word; other CAM operations can set this flag.

The tag flag of an associative word indicates the search result. This flag is set if its associated word has participated in the current search (i.e., it is not empty and its select flag is set) and the word contents matched the conditions of the search; otherwise, the tag flag is cleared.

The word select flag is a one-bit register which allows the user to select a subset of words to participate in a search. If the select flag of a word is set, it can participate in searches; otherwise, it cannot. The select flags can be globally set or they can be loaded from a word's tag flag. Thus the user can use previous search results to narrow the scope of future searches. Such a feature is important to some applications such as database systems.

Like any memory, a CAM must have operations to write data into and read data out of the memory words. Up to five different read/write operations may be needed for a CAM, depending on the applications. These operations are: WRITE.ADDRESS, WRITE.EMPTY, WRITE.MULTIPLE, READ.ADDRESS, and READ.TAG.

The **WRITE.ADDRESS** operation is the most obvious way to write to an associative word in a CAM. In it, the user provides the address of the word to be modified. A decoder uses this address to select the proper word in the CAM to write to, allowing the CAM to be addressed like a conventional RAM. However, some CAM applications may require **address-free** writing, i.e., the ability to write to an empty word without specifying its address (WRITE.EMPTY). In this instance, the **multiple-**

match resolver (MMR) is used to choose an empty word for writing. The MMR is a logic module which accepts an input vector of zeros and ones and produces an output vector consisting of all zeros except for a single one bit. This single bit corresponds to one of the set bits of the input vector. In the **WRITE.EMPTY** operation, the MMR inputs the empty flags of the words and outputs a vector used to select a word for writing. In **WRITE.MULTIPLE** operation, all non-empty words whose tag flags are set are selected for modification. As mentioned earlier, the mask register designates which fields of an associative word get changed.

The **READ.ADDRESS** operation allows the CAM to be accessed like a conventional RAM, i.e., address accessibility. The **READ.TAG** operation is used after performing a search. Here, an associative word whose tag flag is set is read out. Since more than one associative word may have matched the most recent research, the MMR chooses a single word to read from an input vector of flag tags. After reading a word in this fashion, the word's tag flag should be cleared to allow the other tagged words to be read out.

In conventional RAM memory systems, several smaller RAM chips often share the same data and address lines to form a logical RAM with more words than a single RAM chip. This is possible because each RAM has an enable input, and these inputs are controlled so that only one RAM device is active during a read or write operation. Similarly, the proposed CAM has a **CAM enable** input which enables/disables the CAM during addressed read or write operations. This feature allows compatibility with existing RAM chips and will make it easier to interface the CAM to conventional systems.

In such a system where several CAMs share the same data bus, it is necessary to insure that only one CAM will place data on the common data bus during a READ.TAG operation. This can be accomplished by equipping each CAM with a **read enable input and output.** If a disable signal is asserted at this enable input, then the CAM asserts a disable signal on its read enable output and does not respond to any READ.TAG operations. If an enable is asserted at its read enable input, the CAM responds to a READ.TAG operation. In addition, it outputs a disable signal if any of its tag bits are set, and an enable signal otherwise. Using these enable signals, several CAMs can reliably share the same data bus by always asserting an enable signal to the first CAM's read enable input and daisy-chaining all the remaining CAMs' read enable inputs and outputs.

Note that a similar problem (enabling for an operation one of several CAMs connected to the same data bus) occurs if a CAM has a WRITE.EMPTY write ability. In this case, a solution similar to the above (i.e., a write enable input and output) can be used. However, here an enabled CAM will output a disable signal if one of its empty flags (and not tag flags) are set.

VII. MODULAR COMPONENT DESIGNS

This section describes the designs of some of the high-level modules described in the previous section. Some of these designs will be common to all CAMs regardless of their functionality, while others will depend upon the specific functions being implemented in the CAM. For this reason, the exact classification of a module design (generic or function-specific) will be indicated with its description. All the designs are based on CMOS technology.

VII.1 Comparand/Mask Register Module (generic/function specific)

Both of these registers can be constructed out of a 6-transistor static 1-bit storage element discussed in section VII.2 (Figure 6). These storage elements are organized as shown in Figure 7. In this configuration the comparand and mask register are write-only registers. The outputs from both the comparand and mask registers are sent to the signal generation logic. This logic generates the signals needed by the associative word array during the search and maskable write operations. This logic is discussed in the design of the memory storage element and search circuit modules.

VII.2 Memory Storage Element Module (function-specific)

Only static memory elements are considered, mainly because of their simplicity. There are four different storage element designs, each for a different type of CAM write capability: single, non-maskable write; single, maskable write; multiple, non-maskable write; and multiple, maskable write.

The storage element used for single, non-maskable write operations is the six-transistor static memory cell shown in Figure 6. Whenever the memory cell is not being accessed, the word select line is held low. This turns off the pass transistors and the con-

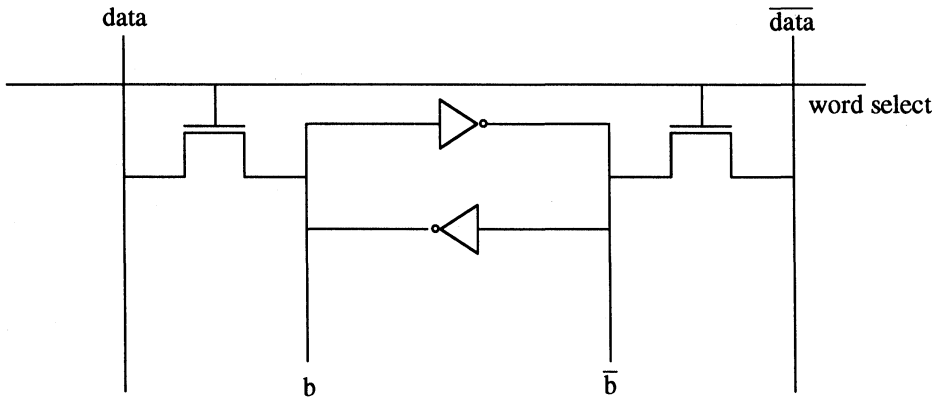

FIGURE 6 1-bit static storage element.

tents of the cell are held by the cross coupled inverters. To read the memory cell, the data and $\overline{data}$ busses are first precharged high. The word select line is then driven high to open both pass transistors. One of the data busses will be slightly discharged due to the low output of one of the two inverters. The exact bus which is discharged depends upon the contents of the cell. The discharge of one of the busses is sensed by column sense amps at the bottom of the memory array [13]. To write to the memory cell, the true value and the complement of the data are driven on the data and $\overline{data}$ busses, respectively. While these busses are driven, the word select line is set high to open the pass transistors. The drive of the superbuffers on the data busses are made large enough to overcome the drive of the cross coupled

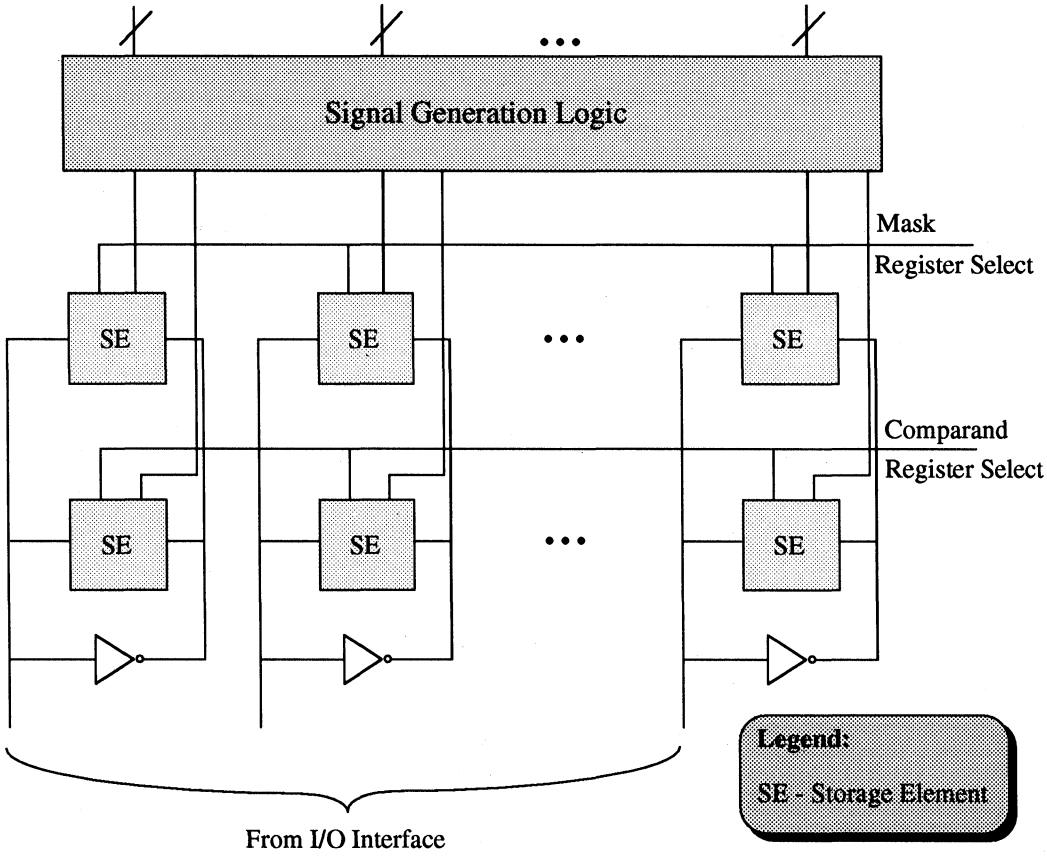

FIGURE 7 Comparand/mask register module.

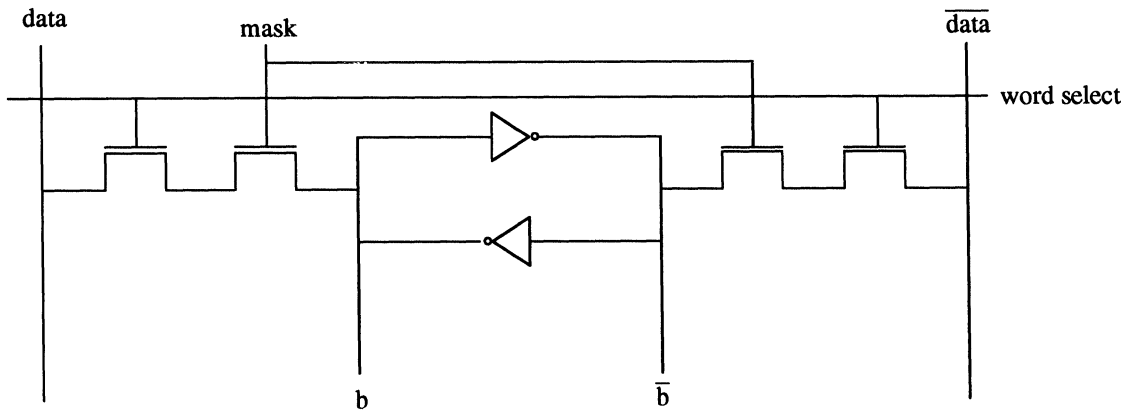

FIGURE 8 Single maskable write memory module.

inverters, which are then driven into the desired state.

To allow maskable write operation, two pass transistors are added to the previous design to arrive at the design found in Figure 8. The write operation is the same as before except that the mask data bus is driven with the mask value for that particular bit. Thus if the mask is zero the mask pass transistors will not operate and the bit will not participate in the write operation. During a read operation both the word select and the mask busses must be held high for the operation to function properly.

In a multiple write situation it is not feasible to make the superbuffer drivers of the data busses large enough to overcome the drive of all the cross coupled inverters participating in the write operation. For this reason, a third design is needed for multiple write situations. One design which allows a multiple, maskable write is shown in Figure 9 and comes from [15] (a variation of this design can be used for a multiple, non-maskable write and is shown in Figure 10). This design requires only 8 transistors, the same as the single, maskable write design. When the module is not being accessed the word write line is held low; this turns on a pMOS pass transistor and completes the feedback loop of the cross coupled inverters so that they hold the stored data. To perform a read operation the data bus is precharged high, the word read line is set high to turn on the pass transistor, and the column sense amps detect whether the data bus is slightly discharged (indicating a stored zero). To perform a write, the data value is driven on the data bus and the word write lines of the memory modules participating in the write are held high to turn off the pMOS pass transistor. Because the feedback loop is broken, the superbuffer data bus driver charges only the capacitive gate of the first inverter and does not need to overcome the drive of the cross coupled inverter pair. The design contains a mask pass transistor if maskable write operations are

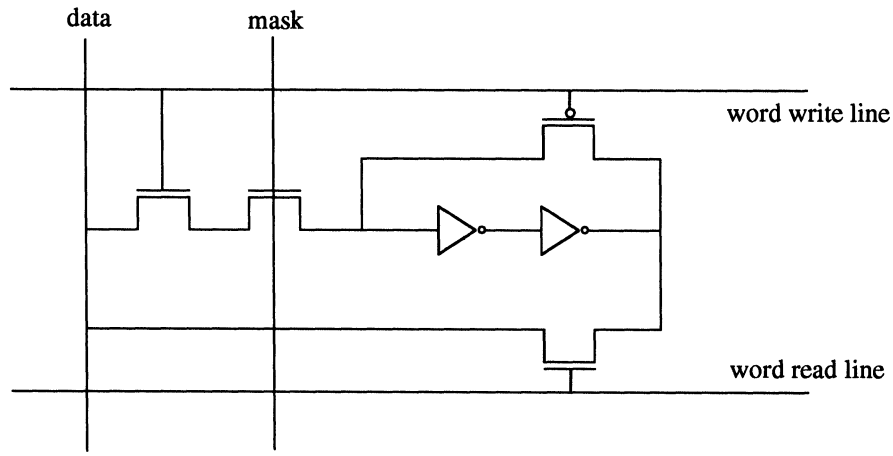

FIGURE 9 Multiple maskable write memory module.

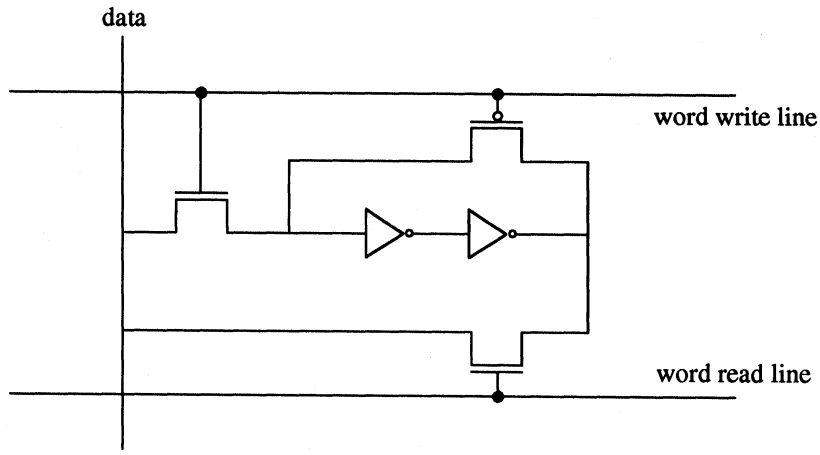

FIGURE 10. Multiple non-maskable write memory module.

needed. Note that a word write line and a word read line are needed instead of a single word select line of the previous designs. One consequence of this design is that the width of the pMOS pass transistor must be carefully controlled so that its threshold voltage allows a zero to be stored properly [15].

VII.3 Search Circuit Module (function-specific)

Paired with every storage element in an associative word, the search circuit module contains the circuitry needed to compare a bit of the associative word with a bit of the comparand register during a search. The actual circuitry used to implement this module depends on whether an equality or θ-search capability is needed.

The design for equality-only search provides high-speed, parallel searching over the entire memory in fixed time (Figure 11). Before a search, the PC line is set low to precharge the MATCH bus. During the search, each bit in each memory word performs an exclusive-nor operation between its content and the corresponding bit in the comparand register to determine if it should conditionally discharge the MATCH bus. Since all bits operate independently, the entire memory is compared in constant time. The precharged design means that only three transistors are needed to perform the actual comparison, adding little area to each memory cell.

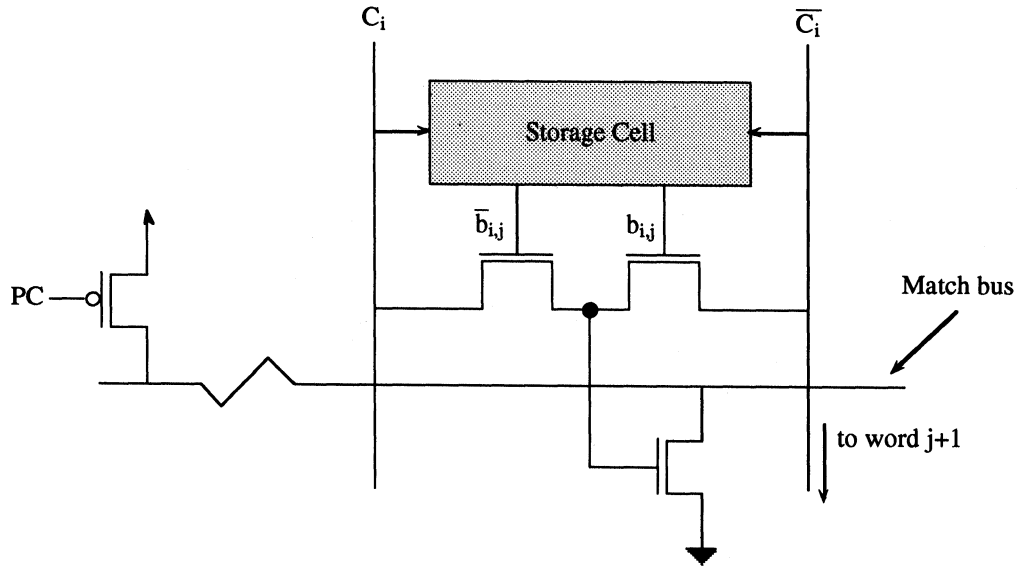

FIGURE 11 EQ-only compare cell.

The θ-search module uses the design presented in section IV and is shown in Figures 3 and 4. The signal generation logic of the comparand/mask register supplies the c_i, $\overline{c_i}$, and FI_i signals needed by the search circuit. As mentioned earlier, the module performs a θ-search in time $O(A)$ where A is the width of the attribute field being searched. This module design requires 15 transistors per bit.

VII.4 Flag Modules (function-specific)

Each word in the associative memory has three 1-bit flags associated with it: the empty flag, the word select flag, and the tag flag. These flags are used to indicate the present status of an associative word. For the purposes of this paper, each flag consists of a D flip/flop (D F/F) driven by a single clock and some input logic. For all three flags the design of this logic depends upon the functions that are included in the CAM.

The empty (EF) module is used to determine whether or not the associative word should be ignored during a search. The input logic to this element must have the following features: the empty flag should be cleared during a write to the word, and the flag should be set during a global RESET. Other desirable operations involving the empty flags might include a destructive read (when a word is read its empty flag is set), selective deletion (words matching the previous search have their empty flags set), as well as other operations. Since each operation sets the empty flags differently, the exact input circuitry for the empty flag depends upon the functionality of the CAM, i.e., it is not identical for all CAMs. The clock signal to the empty flag F/F is provided by the control unit of the CAM.

The word select flag (WS) module determines whether or not its associative word should participate in the current search. The flag is stored in a D F/F, while input logic to the F/F determines how to set and clear the flag. Every CAM should have a global reset command which sets the word select flags of all non-empty words. However, some applications may also require the CAM to have a tag flag copy operation which loads the word select flags of all non-empty words with their corresponding tag flags. This lets the user of the CAM to pick a subset of words to search over. The input logic for the first case is trivial, while the second case is shown in Figure 12. The activation of the clock from the control unit causes the word selects of all the associative words to be loaded with their new values.

The tag flag (TF) module indicates the results of the previous search against the contents of the associative word. For every CAM this logic should allow the tag flag of an associative word to be set to the value of the current search result. However, for some applications it may be beneficial for the tag flag to be set in two other ways:

1) OR the current search result with the previous search result,
2) AND the current search result with the previous search result.

The exact design of the module depends upon the ways in which the flag can be set. Figure 13 shows a design for the tag flag module that allows the flag to be set in all three ways mentioned above. In this figure, the control signals OR and AND indicate which tag set operation should be performed. The CLEAR control signal is always high except after a READ.TAG operation. In this case the clock of the just-read word is activated by the control unit to clear its tag flag. The exact design for the search match logic depends upon the type of searches allowed in the CAM (equality or θ). For an equality search CAM, the search match circuit is merely an exclusive-NOR of a word's EQ bus and the search criteria EQL where:

EQL = 0 search for inequality,

EQL = 1 search for equality

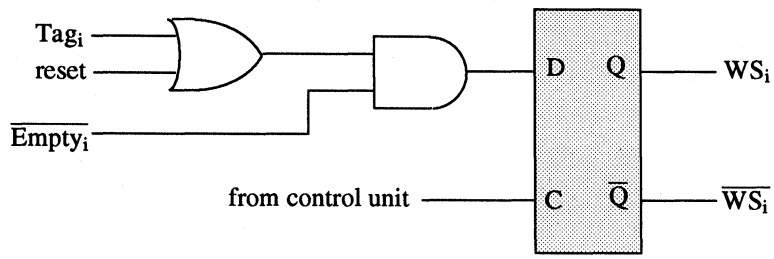

FIGURE 12 Word select module.

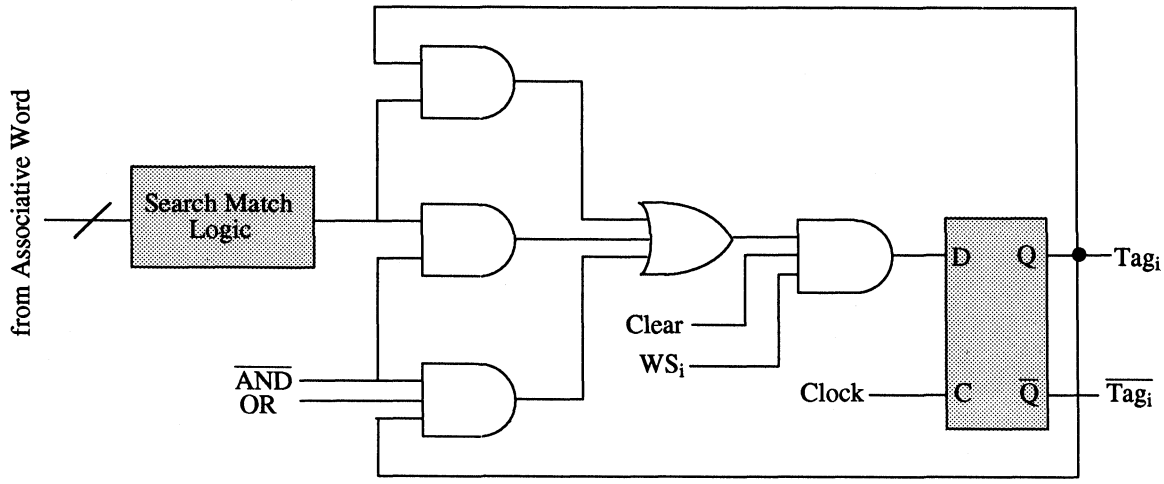

FIGURE 13 Tag flag module.

The search match logic for a θ-search CAM is shown in Figure 14. In this figure E, L, and G are the **equal, less than,** and **greater than** control signals, respectively. The settings of these control signals determine what θ-relation is used during the search. For example, if one is searching for all words greater than or equal to the comparand, both E and G are set. The clock signal, usually provided by the control unit, is activated when the D input should be latched.

VII.5 Decoder Module (generic)

Allowing CAM words to be accessed by their addresses as well as their contents makes it easier to interface the CAM with conventional RAM-based systems. The decoder module takes an externally supplied address and generates a single active-high signal indicating which associative word is selected. This signal is then used to drive the word select line of the appropriate associative word. Brief discussions on decoder design can be found in [22].

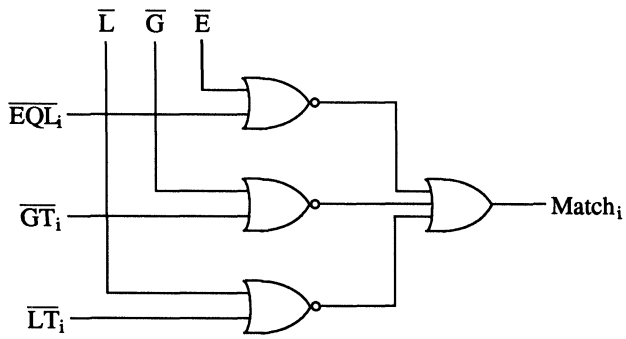

FIGURE 14 Theta-search match logic.

VII.6 MMR Module (generic/function specific)

The multiple-match resolver (MMR) module inputs a vector of binary digits (the tag or empty flags) and outputs a vector containing all zeros except for a single one bit. The design of the MMR is based on the designs found in [1]. The MMR is composed of a tree structure of P-generator blocks. The general structure of the tree is shown in Figure 15 and the design of the P-generator block is shown in Figure 16. The P-generator block will be implemented in static CMOS logic. In the generator block, notice that if $P_{in} = 1$, then P_i will be 1 iff A_i is the first 1 in the A vector, otherwise P_i is 0 (A_{out} is merely the OR of the A vector values). In this design, the P_{in} of the topmost block should be tied to a logic one if any A_i in the A-vector is set. It can be

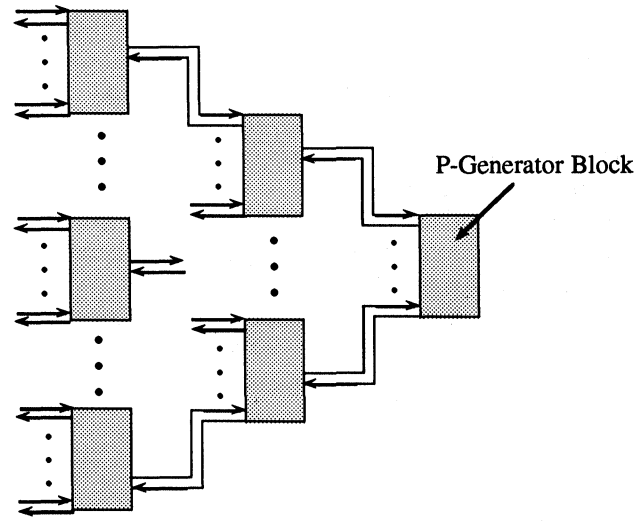

FIGURE 15 P-generation tree.

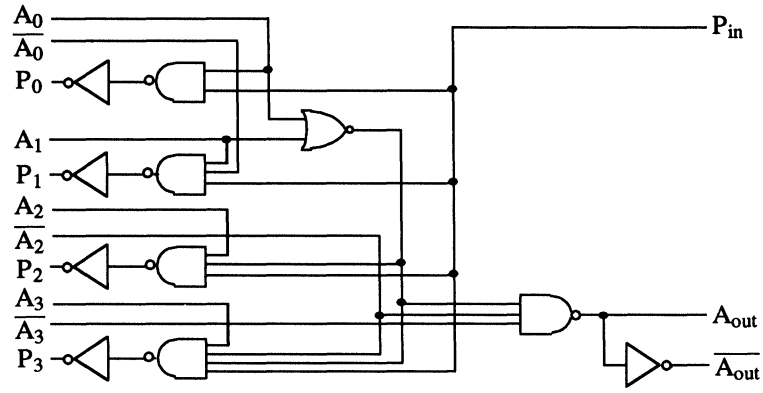

FIGURE 16 P-generator block.

proven that if there are 4^D associative words in the CAM, the MMR constructed from the *P*-generator block of Figure 16 will be D levels deep and consist of $(4^D - 1)/3$ blocks.

For some applications it is desirable to know the address of the word accessed during a WRITE.EMPTY operation or to output only the address (not the contents) of a word which matches the previous search (this is sometimes used in cache memory systems). In both these cases the *P*-vector output of the MMR must be encoded. In other words, if $P_i = 1$ in the *P*-vector, then the encoded address is the binary representation for i. [1] shows a design for an address generator that uses encode logic added to every *P*-generator block. This encode logic generates two outputs, A_{HI} and A_{LO}, and is given by:

$$A_{Hi} = P_3 + P_2$$

$$A_{LO} = P_3 + P_1.$$

For a given level in the tree, all of the A_{HI} and A_{LO} outputs are ORed together to form two bits of the final address. This can be accomplished in pseudo nMOS logic by NORing the values together and complementing the output. However, the width and length of the pull-up pMOS transistor must be chosen carefully to insure that a proper logic low input is generated when only one of the nMOS pull-down transistors is turned on. If pseudo nMOS logic is not allowed, the A_{HI} and A_{LO} outputs must be ORed by an OR tree of static CMOS logic gates. This approach may be larger and harder to lay out than the pseudo nMOS design.

VII.7 Enable Module (generic)

The enable module contains the circuitry that enables/disables various portions of the CAM based on the signals present at the three different enable inputs (the RAM, CAM read, and CAM write enable inputs). It also generates the values placed on the CAM read and write outputs.

Since the RAM enable input serves the same function as the chip select input in conventional RAMs, the design of the control logic using this input signal is not unique to CAMs and will not be discussed further. The CAM read and CAM write enable inputs control whether or not an address-free read or write is performed. In both of these operations the MMR module selects a single associative word. Since the MMR module can be enabled or disabled by controlling the value of P_{in} to the topmost block of the MMR tree, the major function of the CAM read and write enable input circuitry is to control this input of the MMR. If the CAM read input is disabled during an address-free read or the CAM write input is disabled during an address-free write, the P_{in} input to the MMR will be a logic zero; otherwise, P_{in} will be connected to a logic one.

The values of both the CAM read and CAM write enable outputs depend upon the value of the corresponding input enables and whether an associative word was selected by the MMR. If a disable signal is asserted at a CAM enable input or if a CAM input is enabled and the MMR has selected an associative word, a disable signal should be asserted at the corresponding enable output. Assuming that the enable signal is active low and A_{MMR} is the A_{out} of the topmost block of the MMR tree of *P*-generators, the output enable circuitry can be represented by the following equations:

$$CAM_{read\_output} = CAM_{read\_input} + A_{MMR}$$

$$CAM_{write\_output} = CAM_{read\_input} + A_{MMR}.$$

Note that since A_{MMR} is used by this logic, the CAM read(write) output is not valid unless the tag (empty) flags are used as input to the MMR.

VIII. A RELATIONAL DATABASE ORIENTED CAM

This section goes through an example to show how our general CAM architecture can be used to design an application-specific associative memory. A first-order estimate of the size of the sample CAM is also calculated to determine if the design is feasible under current technology. The application chosen is a database machine based on the relational model [8]. The intent is to show the procedure used to build an application-specific CAM using our proposed general architecture.

The first stage of the design process is to determine the exact functionality needed by the CAM. For our example we decided to stress six commonly used relational operations: project, select, join, update, insert, and delete. We determined that to perform these operations efficiently the following features are needed:

1. maskable WRITE.MULTIPLE capability,
2. θ search capability,
3. accumulative search: capability to OR or AND the current search result with the previous tag flag before storing it in the tag flag,
4. selective deletion: capability to OR the empty flags with their corresponding tag flags and storing the results back in the empty flags,
5. search subset selection: capability to copy the tag flags into their corresponding word select flags,
6. WRITE.EMPTY capability,
7. READ.TAG capability,
8. word select and empty flag reset capability, and
9. READ.ADDRESS and WRITE.ADDRESS capability.

Figures 7, 8, and 9 are common to virtually every CAM. [19] contains an associative memory algorithm to perform the project operation which, if slightly modified, will operate on a CAM with feature 5. The select operation can be performed with features 2 and 3. [7] shows how to implement a join on a CAM with feature 2. Finally, updates and deletions use features 1 and 2, and insertions use features 2, 4, and 6.

We will assume that the associative memory consists of n words, each m bits wide. Ratioless CMOS complementary logic (including complex gates) will be used wherever possible due to its design simplicity and efficient layout style. A first order area estimate can be calculated by multiplying the total number of transistors by the size of one transistor [22]. Minimum sized transistors (i.e., 2λ by 2λ) will be used to minimize space. The reduced speed caused by using these transistors should be offset by the parallelism inherent in the CAM.

Each bit of the mask and comparand register can be stored in a 6 transistor static storage element (Figure 6). The signal generated by logic for this module needs only to generate c and FI_i signals needed by the θ-search circuitry. Each of these signals can be generated by a separate 4-transistor CMOS NOR gate. The total number of transistors needed for this module is $24m$, making the estimated area for this module $96m\ \lambda^2$.

The memory element from Figure 10 will be used. Since this contains 8 transistors, the total size of these elements in the CAM will be approximately $32mn\ \lambda^2$.

The θ-search element from Figure 3 is used. This element uses 15 transistors per bit, resulting in an estimated size of $60mn\ \lambda^2$.

Each of the three flag modules consists of a D F/F and some input logic. The design of the D F/F is shown in Figure 17 and consists of 18 transistors. In

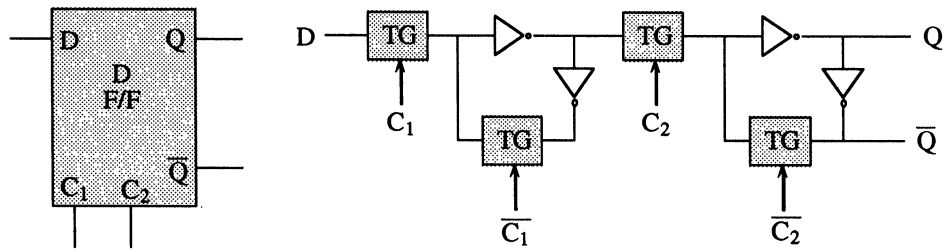

FIGURE 17 A master/slave D flip/flop.

this figure, TG stands for a 2-transistor CMOS transmission gate. Here, only the input to the nMOS transistor in the TG is shown; the pMOS transistor in the TG should receive the complement of this signal. Data is latched to the output of the F/F when the clock signal goes high.

The input logic to the empty flag *D* F/F is represented by: D_i = ORTag$_i$ + Empty$_i$ + EReset. EReset is held high to globally reset the empty flags, and OR is asserted to selectively delete tagged words from the memory. This input logic can be represented by a single complex gate containing 8 transistors. Combined with the 18 transistors needed for the *D* F/F, each empty flag requires 26 transistors. Thus the total estimated area for the empty flags is $104n\ \lambda^2$.

As shown in Figure 12, the input logic for the word select flag *D* F/F is given by: $D_i = \overline{\text{Empty}_i}$ (Tag$_i$ + WReset). WReset is asserted to select all non-empty words for searching and deasserted to perform a tag flag copy. Using a 6-transistor complex gate for the input logic and the 18-transistor *D* F/F mentioned earlier, each word select flag has 24 transistors. The total estimated area for the *n* word select flags is $96n\ \lambda^2$.

The input logic of the tag flag *D* F/F is divided into two parts: the search match logic and the tag set logic. The search match logic (shown in Figure 14) is given by: Match$_i$ = G_iGT + L_iLT + E_iEQ. Its corresponding complex gate requires 18 transistors. The tag set logic, shown in Figure 14, is: D_i = Clear · $\overline{WS_i(\text{Tag}_i \cdot \text{Match}_i + \overline{\text{AND}} \cdot \text{Tag}_i \cdot \text{OR} + \text{Match}_i \cdot \overline{\text{AND}})}$. A complex gate for this function would use 18 transistors. Each tag flag uses both complex gates plus a *D* F/F for a total 54 transistors. Thus the estimated area for the tag flags is $216n\ \lambda^2$.

In our model a tree of 2:4 decoder elements with active high enables and outputs is used to construct the decoder. This structure was chosen for design simplicity, not for efficient use of area. The logic diagram of the 2:4 decoder is shown in Figure 18. This element can be represented by a complex gate containing 30 transistors. It can be proven that if there are $n = 4^D$ words in the associative memory then the decoder tree can be built out of $(n - 1)/3$ decoder elements. The estimated area for the decoder module is $40(n - 1)\ \lambda^2$.

The MMR module can be built out of *P*-generator blocks shown in Figure 16. Each of these blocks requires 44 transistors if built in CMOS static logic. As in the decoder, if there are $n = 4^D$ associative words in the CAM the MMR module would use $(n - 1)/3$ of these blocks. This makes the estimated area for the MMR module at least $59n\ \lambda^2$.

A first order estimate of the size required for the CAM can be calculated from the area estimates:

$$\text{Area} = (92mn + 515n + 96m)\lambda^2$$

However, note that this analysis did not consider the area required for control logic, enable logic, superbuffer drivers, etc. We estimate that this logic should require no more than 5000 transistors. In addition, layout inefficiencies due to routing were not considered. To compensate for this the area estimate is multiplied by a factor of 60. For a 1024 words each 32 bits long (n = 1024, m = 32) the estimated geometric area using $\lambda = 1\ \mu m$ is 213.9 $(\text{mm})^2$, or approximately $14.6mm \times 14.6mm$. Very large associative memories (e.g., 1 Mword) require a large amount of silicon area, but feasible by using wafer-scale integration technology.

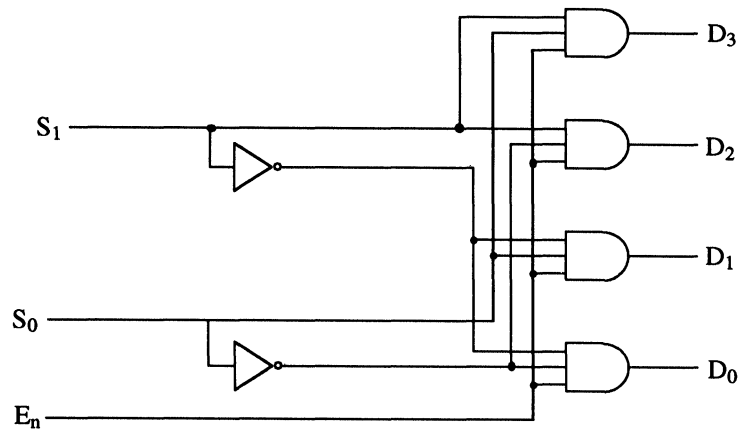

FIGURE 18 2:4 decoder with enable. a) Accuracy. b) Timing.

IX AUTOMATED DESIGN TOOL

Section V enumerated some of the advantages of a modular design, apart from those properties, a modular organization also paves the way for a wider use of design automation. The term automation refers to the use of mechanized means in the execution of the repetitive steps of a process. In the case of modular designs, the process of choosing appropriate modules to fit an already outlined system structure is a well-suited task for automation. Naturally, in order for an automated design process to be useful, the solutions have to be rated with respect to certain criteria, e.g., cost and performance factors.

We are interested in developing an automated tool that would help designers in putting a modular associative chip together. Such a tool would appropriately select versions of components from a design library which: i) are optimized with respect to a cost function, and ii) satisfy the constraints and functionality set by the user. Furthermore, this tool should allow the designer to explore alternative solutions and inform him/her of the realizability of such a design under the imposed constraints.

IX.1 Genetic Algorithms as an Optimization Tool

As their name implies, genetic algorithms were conceived as models mimicking adaptation in nature. The term adaptation refers to the ability of different systems to adapt to their changing environment in ways that would prove beneficial. All the information about an individual living organism and its species is carried in its genetic material kept in each one of its cells. The genetic information about a species is not carried merely in a single individual, but collectively within the genetic material of the individuals in a population. Each characteristic is defined by a portion of the DNA code called a gene. According to the theory of adaptation, as individuals reproduce, they mix their genetic material and produce individuals with new gene combinations and thus new sets of characteristics. Those individuals who are best adapted to live in the current environment will have the greatest chance to leave offspring, and pass on their genetic material—the principle of the survival of the fittest.

Genetic algorithms represent an attempt to simulate the natural process of selection and its optimizing properties. For a given optimization problem, solutions can be represented by strings which encode all their characteristics. The genetic algorithm begins with an initial population of individuals, usually chosen at random from the entire search space, i.e., all the possible individuals. Each individual, i, in a generation of parents is marked for its performance with a fitness value, f_i. This value will determine the number of offsprings that the individual will be allowed to have. The higher an individual scores, the more chances it will be given to reproduce. On the other hand, those that did not favor as well will be less instrumental in the making of future populations.

To incorporate genetic material from both parents in the resulting offsprings, a crossover operation is performed. The crossover point is the point at which the strings of genetic code of both parents will be split. The first portion of the first parent string will be complemented by the second portion of the second parent string, while the second portion of the first parent string will be appended to the first portion of the second parent string. The crossover operation mixes genetic information from two parents. Since the parents have been chosen because of their good performance, there is a good possibility that their offspring will inherit the good characteristics from both its parents and discard the bad ones. Naturally, it is just as likely that the opposite is accomplished. However, due to the selection process, unsuccessful offsprings are not included into future generations, whereas successful ones are.

Just like natural systems, genetic algorithms draw their power from the great diversity of their population. The more diverse the initial population is, the more genetic material is available and the more effective the search is. In cases where the initial population is dominated by a few "fit" individuals, the genetic information carried by "weaker" individuals is lost too early in the search. This may result in solutions that are merely local maxima of the optimization function. As a result, in our implementation, proper steps were taken to avoid such **genetic disasters.**

Given the standard feature of genetic algorithms, their actual implementation can be a generalized set of procedures designed to operate on strings of information encoding the characteristics of an individual. The specifics of the encoding scheme for each implementation can be hidden in the fitness evaluation function and the splitting of strings for the crossover operation. Our implementation utilizes such generalized procedures. However, modifications were made to allow the crossover and mutation operators to work properly. String decoding and fitness evaluation was redefined to fit the problem at hand.

In the case of genetic algorithms, the search is concluded when convergence of the population is

reached. Convergence implies that the majority of individuals left in the population after a number of generations are essentially the same. From then on, there is no point in carrying on with the genetic algorithm, since all possible combinations of individuals produce no new individuals. Therefore, unless mutation probability is very high, there is no new genetic material in the current population that can be explored.

We have implemented the genetic algorithm described, as a means to finding a solution to our optimization problem. To determine suitable values for the aforementioned parameters, a series of tests was run on a sample 8 × 8 table, with probability of mutation (P_M) ranging from 0.01 to 0.05, probability of crossover (P_C) ranging from 0.2 to 1.0, and population size of 80. The algorithm was run 30 times and statistics regarding its accuracy (i.e., the number of times the algorithm was successful in finding the optimal solution) and speed (i.e., the time to reach convergence) were observed. Figure 19 depicts the results.

X. CONCLUSIONS AND FUTURE PLANS

Associative memories can significantly improve the performance of many applications, including memory management, database machines, and fifth-generation computers. With the advent of VLSI tech-

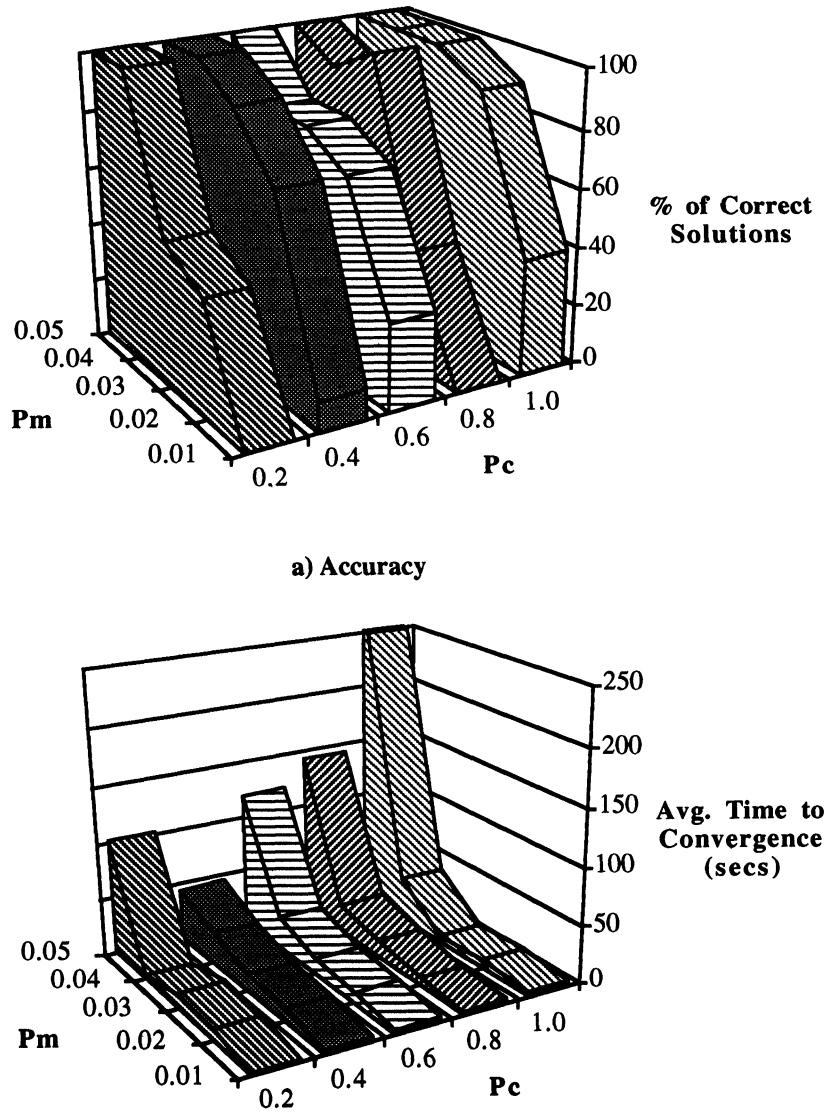

FIGURE 19. Performance measures vs. probability of convergence and probability of mutation (population size = 80).

nology, high-performance fully parallel CAMs are now feasible and we surveyed several proposed special-purpose designs in this paper. In addition, an improved Θ-search cell was presented. To meet the varying requirements of different applications, a general, modular CAM organization is needed to simplify the design procedure of specialized, cost-effective associative memories. We then introduced a high-level CAM architecture consisting of modular components, explained the function of each module, and presented the design of several of the modular components. To illustrate the design procedure of a special purpose CAM using our proposed organization, we considered a sample associative memory suitable for performing relational database operations. The designs for most of the major modules were presented and a first order size estimate of the CAM was derived. We determined that for a CAM with those particular functions, a 1K associative memory of 32-bit words has an estimated geometric area of $14.6mm \times 14.6mm$. This is within the current capabilities of technology.

We are currently researching ways to further improve our general CAM organization. Many of these include adding additional features to the CAM. Some of the features are simple to incorporate in a CAM, e.g., counting the number of matches during a search or allowing the tag and word select flags to be read out. Other features are not as trivial. For example, we have been examining how to incorporate some degree of fault tolerance in the CAM and make it a standard feature. This may be done by adding a new flag, called a faulty flag, to every associative word. This flag indicates whether the word is faulty and inhibits certain operations performed on the word if it is set. Another research area is extending the logical width of an associative word. This is particularly useful when searching over items (e.g., tuples in a relational database) that are larger than the physical word size of the associative memory.

Our future plans also include designing an application-specific associative memory using our proposed organization, implementing it in full custom CMOS VLSI, and testing its performance. This would allow us to determine the best designs for several of the modular components. Since the exact size of many of the modules will then have been determined, better size estimates for other application-specific CAMs could be obtained. A general design for the control unit of the CAM might also be discovered during the design process. Finally, a successful prototype might further encourage the general use of application-specific CAMs.

References

[1] G.A. Anderson, "Multiple Match Resolvers: A New Design Method," *IEEE Transactions on Computers,* pp. 1317–1320, December 1974.

[2] E.W. Davis, "STARAN Parallel Processor System Software," *National Computer Conference,* pp. 17–22, 1974.

[3] K.E. Grosspietsch, "Associative Processors and Memories: A Survey," *IEEE Micro,* Vol. 12, No. 3, pp. 12–19, 1992.

[4] J. Hayes, *Computer Architecture and Organization,* New York: McGraw-Hill Book Company, 1988.

[5] F. Herrmann and Sodini, "A Dynamic Associative Processor for Machine Vision Applications," *IEEE Micro,* Vol. 12, No. 3, pp. 31–41, 1992.

[6] A.R. Hurson and B. Shirazi, "Associative Memories: Has Their Time Come? Applications and VLSI Complexities," *Proceedings of the 20th Annual Hawaii International Conference on Systems Sciences,* pp. 284–292, January 1987.

[7] A.R. Hurson, S.H. Pakzad, D.B. Shin, and L.L. Miller, "A Reconfigurable Backend Database Machine," *Journal of Parallel and Distributed Computing,* Vol. 11, pp. 37–50, 1991.

[8] A.R. Hurson, L.L. Miller, S.H. Pakzad, M.H. Eich, and B. Shirazi, "Parallel Architectures for Database Systems," *Advances in Computers,* Vol. 28, pp. 108–151, 1989.

[9] A.R. Hurson and P.M. Miller, "A 16-K bit Θ-Search Associative Memory," *IEEE Micro,* Vol. 13, No. 2, pp. 59–65, 1993.

[10] S.R. Jones, I.P. Jalowiecki, S.J. Hedge, and R.M. Lea, "A 9-k bit Associative Memory for High-Speed Parallel Processing Applications," *IEEE Journal of Solid-State Circuits,* Vol. 23, No. 2, pp. 543–548, April 1988.

[11] H. Kadota, J. Miyake, Y. Nishimichi, H. Kudoh, and K. Kagawa, "An 8-k bit Content-Addressable and Reentrant Memory," *IEEE Journal of Solid-State Circuits,* Vol. SC-20, No. 5, pp. 951–957, October 1985.

[12] T. Kohonen, *Content-Addressable Memories,* New York: Springer-Verlag, 1980.

[13] A. Mukherjee, *Introduction to nMOS & CMOS VLSI Systems Design,* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[14] J. Nahanuma, T. Ogura, S.-I. Yamada, and T. Kimura, "High-Speed CAM-Based Architecture for a Prolog Machine (ASCA)," *IEEE Transactions on Computers,* Vol. 37, No. 11, pp. 1375–1383, November 1988.

[15] T. Ogura, S.-I. Yamada, and T. Nikaido, "A 4-k bit Associative Memory LSI," *IEEE Journal of Solid-State Circuits,* Vol. SC-20, No. 6, pp. 1277–1282, December 1985.

[16] T. Ogura, S.-I. Yamada, and J. Yamada, "A 20K bit CMOS Associative Memory LSI for Artificial Intelligence Machines," *Proceedings of the IEEE International Conference on Computer Design,* New York, NY, pp. 574–577, October 1986.

[17] E.J. Oliver and P.B. Berra, "RELACS, A Relational Associative Computer System," *Fifth Workshop on Computer Architecture for Non-Numeric Processing,* CA, pp. 106–114, June 1980.

[18] C.R. Petrie and A.R. Hurson, "A VLSI Join Module," *VLSI Systems Design,* Vol. 9, No. 10, pp. 46–58, 84, October 1988.

[19] D.E. Shaw, "A Relational Database Machine Architecture," *Fifth Workshop on Computer Architecture for Non-Numeric Processing,* CA, pp. 84–95, June 1980.

[20] C.D. Stormon, M.R. Brule, J.V. Oldfield, and F.C.D.F. Riberio, "An Architecture Based on Content-Addressable Memory for Rapid Execution of Prolog," *Proceedings of the Fifth International Conference on Logic Programming,* pp. 1448–1473, 1988.

[21] J. Wade and C. Sodini, "A Ternary Addressable Search Engine," *IEEE Journal of Solid State Circuits,* Vol. 24, No. 4, pp. 1003–1013, 1989.

[22] N. Weste and K. Eshraghian, *Principles of CMOS VLSI*

Design—A Systems Perspective, Reading, MA: Addison-Wesley Publishing Company, 1985.

[23] S.S. Yau and H.S. Fung, "Associative Processor Architecture—A Survey," *ACM Computing Survey,* Vol. 9, No. 1, pp. 3–27, March 1977.

Biographies

A.R. HURSON is a Computer Engineering Faculty at The Pennsylvania State University. His research for the past 12 years has been directed toward the design and analysis of general as well as special purpose computer architectures. He has published over 120 technical papers in areas including computer architecture, parallel processing, database systems and database machines, dataflow architectures, and VLSI algorithms. Dr. Hurson served as the Guest Co-Editor of special issues of the *IEEE Proceedings on Supercomputing Technology,* the *Journal of Parallel and Distributed Computing on Load Balancing and Scheduling,* and serving as the Guest Co-Editor of the *Journal of Integrated Computer-Aided Engineering on Multidatabase and Interoperable Systems.* He is the co-author of the *IEEE Tutorials on Parallel Architectures for Database Systems and Multidatabase Systems: An Advanced Solution for Global Information Sharing.* He is also the co-founder of the *IEEE Symposium on Parallel and Distributed Processing.*

Professor Hurson has been active in various IEEE/ACM Conferences and has given tutorials for various conferences on database management systems, supercomputer technology, data/knowledge-based systems, scheduling and load balancing, and parallel computing. He is a member of the IEEE Computer Society Press Editorial Board and a member of the IEEE Distinguished Visitors Program.

S. PAKZAD is an assistant professor of Computer Engineering at The Pennsylvania State University. Her research interests include parallel processing, interconnection networks, database systems, neural networks, and fault-tolerant multicomputers. Professor Pakzad has published over 50 technical papers in the areas of databases, neural network processing, and fault-tolerant interconnection networks. Dr. Pakzad is the co-author of the *IEEE Tutorials on Parallel Architectures for Database Systems and Multidatabase Systems: An Advanced Solution for Global Information Sharing.*

Pakzad received her M.S. in Computer Science from the University of Iowa and her Ph.D. in Computer Science from the University of Oklahoma.