A Programmable Dynamic Interconnection Network Router with Hidden Refresh

José G. Delgado-Frias, *Senior Member, IEEE,* Jabulani Nyathi, and Douglas H. Summerville, *Member, IEEE*

***Abstract*— A VLSI implementation of a programmable pipelined router scheme for parallel machine interconnection networks is presented in this paper. The implementation is based on a dynamic content-addressable memory (DCAM) that supports unique bit masking per entry. The number of required DCAM entries is extremely small; it is of the same order as the node degree (output ports). This, in turn, makes it possible to implement a dynamic content-addressable memory in order to reduce the physical size of the system. A DCAM is implemented with only six and a half transistors (one transistor is shared by two cells). We have provided circuitry and arranged timing to achieve refreshing of the stored data in a hidden fashion. In addition to the DCAM, we have incorporated a fast priority scheme that allows only one entry to be selected. The router executes routing algorithms in 1.5 clock cycles, this being the fastest approach for flexible routers. The prototype router has 24 entries, and is able to sustain a throughput of one routing decision per cycle.**

***Index Terms*—Content addressable memory, dynamic circuits, matching with ternary digits, parallel comparison, pipelining, routing algorithm execution, ternary digit logic.**

I. Introduction

PARALLEL computer organizations have been proposed and developed to provide the performance required in a number of applications [1], [5], [7]. A parallel system consists of a number of processing elements (PE's) that are arranged in a configuration that is characterized by an interconnection network. Communication between PE's is accomplished by means of the interconnection network, which should not only provide a short path, but also accommodate the communication needs of the application. A number of interconnection network topologies have been proposed and used [4]; each network has features that make it suitable for a set of applications and algorithms [2], [10].

An interconnection network node has a router that provides a means of handling messages on the network. The router receives, forward, and delivers messages as well as controlling message flow through the network. The router system transfers messages from its input ports to the proper output ports based on a routing algorithm. A crucial component of any large-scale parallel machine is the routing algorithm [9]. The router has to be able to accommodate the routing requirements of an interconnection network topology. These requirements may include: routing algorithm execution, expeditious determination of the destination port, flexibility to accommodate modifications to the network, support for a large number of interconnection networks, deterministic communication path, and programmability. The VLSI CAM-based router scheme has been designed to support routing algorithms that are used in most processor-based router schemes [11]. Using routing algorithms to determine the output port requires consideration of the following two issues [12].

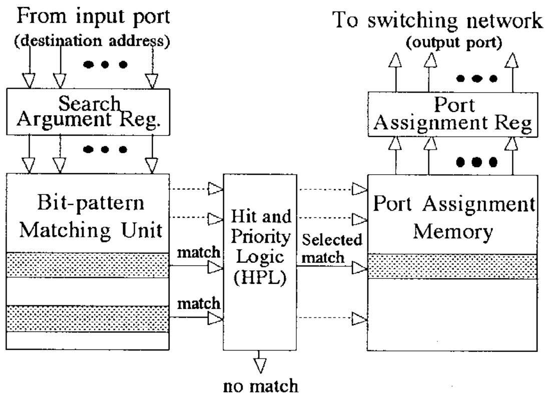


Fig. 1. CAM-based router organization.

- *Some addressing bits need be ignored depending on the current and destination nodes.* A routing algorithm examines the status of addressing bits that are of importance to determine the output port. The bits that are taken into account depend on current node position in the network topology, destination address, routing algorithm, and output port priority. For instance, in a hypercube topology, when routing to the higher unrouted dimension (e-cube algorithm), the bits of lower dimensions are not considered [12]. In this case, the lower dimension bits are not considered since they do not affect a decision of taking a route in a high-dimension channel.
- *Output port alternatives need be prioritized.* This priority is used to favor a port that would yield a short path. This feature ensures deterministic routing algorithm execution.

Our CAM-based router provides a bit-pattern matching mechanism that allows all the alternative paths to be considered in parallel. Fig. 1 shows the proposed router. The modules along with the matching mode of operation for this router are briefly described below.

1) *Search Argument Register*: The destination address is passed to the router by an input port. At this time, the search argument register latches this address, which serves as input to the following stage.

Manuscript received August 27, 1996; revised December 9, 1997 and June 4, 1998. This paper was recommended by Associate Editor K. Antreich.

J. G. Delgado-Frias and J. Nyathi are with the Department of Electrical Engineering, State University of New York, Binghamton, NY 13902-6000 USA (e-mail: delgado@binghamton.edu).

D. H. Summerville is with the Department of Electrical Engineering, University of Hawaii, Honolulu, HI 96822 USA.

Publisher Item Identifier S 1057-7122(98)07721-6.

2) *Bit-Pattern Matching Unit*: The bit patterns for a given interconnection network node are stored in this unit. This unit performs a comparison between the destination address and the current address (which is specified by the bit patterns). All of the bit patterns that match the input address are passed to the following stage.
3) *Hit and Priority Logic (HPL)*: If more than one pattern has been found to match, the hit and priority logic selects only one match. The other matches are ignored. If no pattern matches with input (due to an unknown address destination), the HPL sets a no match signal. The selected match pointer is passed to the following stage.
4) *Port Assignment Memory*: The output port assignments are stored in this memory. The address where the assignment is read is specified by a pointer from the hit priority logic. This assignment is passed to the port assignment register.
5) *Port Assignment Register*: The output port assignment is stored in this register to be used by the interconnection network.

In addition to the matching mode of operation, there is a program mode in which the bit patterns and port assignments are stored into memory. During the program phase, the match and HPL logic are ignored, and an address decoder (not shown) is used to select the word in both memories that is to be written.

The CAM design should include a mechanism to mask off any set of bits at each entry. This mechanism selectively ignores bits that have no relevance in determining a routing path. In order to discard bits within a CAM entry, it is necessary to have a ternary condition for each bit comparison. Thus, each CAM cell needs to store this ternary condition which consists of binary logical values (ZERO and ONE) as well as don't care (X). A set of these ternary conditions per entry constitutes a bit pattern. Each of the bit patterns is compared in parallel with the destination address passed by the search argument register [12].

The hit and priority logic implements a prioritization of the match lines from the matching unit. A priority function is required to ensure deterministic execution of the routing algorithms, and to select the output port that provides a short path. The priority function enables one of the match lines from the matching unit to address the port assignment memory. The hit and priority logic has been designed using a dynamic approach.

The proposed dynamic CAM-based router has been designed and implemented using CMOS technology. This implementation is a vast improvement over our previous static CAM router [3]. By having a dynamic CAM router, we have reduced the total transistor count significantly without losing the router functionality and performance.

This paper has been organized as follows. Sections II and III outline the circuit design used in the full custom CMOS VLSI implementation of the CAM-based router. In Section II, the circuit design of the matching unit is presented in detail. In Section III, the design and implementation of the hit and priority logic as well as the port assignment memory are introduced. The timing of the router operation as well as the results of the CMOS implementation are given in Section IV. Section V presents an application of the proposed router approach: hypertree routing. Concluding remarks are provided in Section VI.

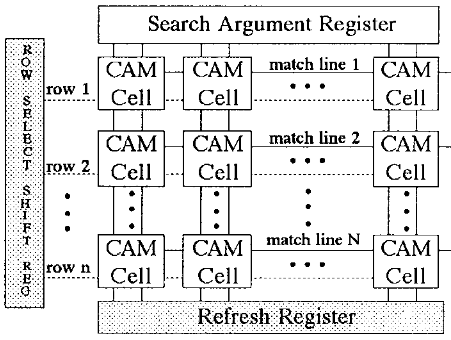


Fig. 2. Matching unit organization.

II. Matching Unit Implementation

In our VLSI implementation, the bit-pattern matching unit has been realized in the form of a dynamic content-addressable memory (DCAM) design. The DCAM design includes per entry unique bit masking to provide the parallel evaluation of all of the bit patterns on the input data. The design uses dynamic storage cells to hold the matching data condition. This condition requires a ternary digit (with values 0, 1, and X) per comparison bit. The refreshing requirement of the dynamic cells has been made transparent by interleaving the refresh operation with the match operation to maintain the high level of performance achieved using static storage [3]. In addition, the hardware resources are shared at different times during a clock cycle in a nonconflicting manner. The resulting DCAM cell is smaller than the static cell while maintaining the same level of performance and functionality.

A. Match Unit Organization and Cell Design

The organization of the matching unit is shown in Fig. 2. The CAM cells are dynamic circuits that perform the bit comparison between the stored patterns and the argument (destination address) sent by the search and argument register (SAR). The comparison is done by means of an XOR-like operation. Once a match with all of the patterns has been performed, the condition of the match lines is latched; this is needed to accommodate the pipeline requirements. In order to provide programmability and refresh to the dynamic CAM (DCAM) cells, two logic circuits (shown as gray blocks in Fig. 2) are used: row select shift register and refresh register. The row select shift register provides signals (row 1, row 2, $\cdots$, row n) to select a row in the CAM array to either read from or write to. This pointer is used for the programming and refreshing modes. In the refreshing mode, the refresh register holds the data and passes back this information.

In the design of the DCAM cell, we have stressed not only transistor count reduction, but also performance and hidden refresh issues. The circuit of a single DCAM cell, shown in Fig. 3, consists of six and a half transistors; transistor T_E is shared by two cells. This circuit shares some similarities

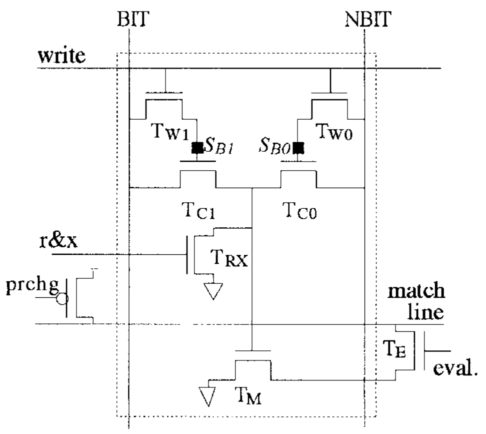


Fig. 3. CMOS circuit for a single DCAM cell.

TABLE I
REPRESENTATION OF STORED DATA IN A DCAM CELL

S_{B1}	S_{B0}	Stored Data Value
0	0	X (Don't Care)
0	1	1 (ONE)
1	0	0 (ZERO)
1	1	*Not Allowed*

with the dynamic CAM circuit proposed by Wade and Sodini [14]. However, our circuit provides much shorter matching and reading delays and accommodates hidden refresh.

The read (denoted as $r\&x$ in Fig. 3), write, evaluation, and match line signals are shared by the cells in a word, while the BIT and NBIT lines are shared by the corresponding bit in all words of the matching unit. The design uses a precharged match line to allow fast as well as simple evaluation of the match condition. This condition is represented by the DCAM cell state which is set by the S_{B1} and S_{B0} node status. This state is stored in the gate capacitance of the transistors T_{C1} and T_{C0}. The possible DCAM cell stored values are listed in Table I.

The DCAM cell performs a match between its stored ternary value and a bit input (provided by BIT and NBIT which represent a bit and its opposite value, respectively). The match line is precharged to a logic 1; thus, when a no match exists, this line is discharged by means of transistor T_M. To read the contents of the DCAM cell, transistor T_{RX} is set on. If the stored bit in S_{B1} (or S_{B0}) is a "1," then the BIT line (or NBIT line) will be set to 0. If both S_{B1} and S_{B0} are "0" (don't care condition), transistor T_{RX} sets a "0" at the gate of T_M; this, in turn, serves as refresh for the don't care condition at transistor T_M. A more comprehensive description of this cell operation is provided in the following subsections.

B. Match Operation

The match operation mode involves comparing the input data to the patterns stored in the DCAM and determining if a match has been found. During match operation, the input data to be compared with the stored data are presented on the BIT line and their inverse value on the NBIT line. Before the actual matching of these two values is performed, the match line is precharged to V_{DD}, which indicates a match condition.

TABLE II
LOGIC AND TRANSISTOR STATUS TABLE FOR THE MATCH OPERATION

Stored Value	DCAM bits		input		transistor status			Match Condition
	S_{B1}	S_{B0}	BIT	NBIT	T_{C1}	T_{C0}	T_M	
0	1	0	0	1	ON	OFF	OFF	match
0	1	0	1	0	ON	OFF	ON	no-match
1	0	1	0	1	OFF	ON	ON	no-match
1	0	1	1	0	OFF	ON	OFF	match
Don't Care	0	0	X	X	OFF	OFF	OFF	match

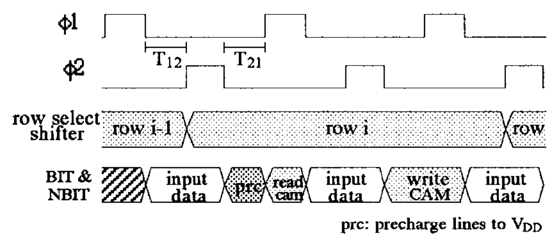


Fig. 4. Refresh READ and WRITE timing.

The matching of the input data and the stored data is performed by means of an exclusive-or operation which is implemented by the two transistors (T_{C1} and T_{C0}) that hold the stored value. The match condition under different inputs and stored data is shown in Table II. When the stored value is either 0 or 1, the exclusive-or circuit sets a path from BIT or NBIT, respectively, to the gate of transistor T_M. This allows the exclusive-or circuit to pass either 0 or 1 to that gate. When there is a match between the two values, a 0 is passed; this condition prevents transistor T_M from discharging the match line. On the other hand, when there is no match, T_M's gate is set to 1. This, in turn, sets a path to discharge the match line when the evaluation signal is set high. If the stored value is set to "don't care," transistors T_{C1} and T_{C0} are off, disconnecting the BIT and NBIT lines. For the "don't care" value, transistor T_{RX} sets the gate of transistor T_M to zero; this, in turn, prevents the cell from discharging the precharged match line.

C. Hidden Refresh Operation

The refresh operation occurs at a portion of the clock cycle when the BIT and NBIT lines are not needed for the match operation (shown as the shaded areas in Fig. 4). This makes the refresh operation transparent to the match operation. The refresh circuitry consists of the row select shift register and the refresh register (shown in Fig. 2). The row select shift register points to the matching unit row that is being refreshed. To prevent the refresh operation from extending the cycle time, the operation has been split into two phases: read stored data and write back data. Fig. 4 shows the timing for the refresh read and write along with the match operation. We have used a two-phase clock (ϕ_1 and ϕ_2); the times when both clocks are 0 are called T_{12} and T_{21}.

A DCAM row, selected by the row select shift register, is read and stored at the refresh register. The read operation is similar for both bits of the stored value (S_{B1} and S_{B0} shown in

TABLE III
KILL, PASS, AND Priority$_{\text{out}}$ GENERATION

Match	Kill	Pass	Priority$_{\text{out}}$
0	0	1	Priority$_{\text{in}}$
1	1	0	0

Fig. 3). Since the data stored in the cell are dynamic, the read operation must sense the data stored at S_{B1} and S_{B0} without weakening them. This is accomplished by transistor T_{RX} in Fig. 3. To ensure the proper reading operation, the BIT and NBIT lines are precharged just before reading takes place (at time T_{21}) as shown in Fig. 4. At ϕ_1, the inverse value of each bit is read. For instance, if a 1 is stored at S_{B1}, this causes the BIT line to be discharged; on the other hand, a stored 0 prevents the discharge of the line. The refresh register inverts the data before setting the BIT line at the write-back phase. At the write-back phase, the value data are passed to the BIT and NBIT lines at time T_{21} before the actual writing occurs (at ϕ_1). This, in turn, helps to reduce the delay on writing into the cell. This is particularly important when writing a 1 which is done through an n-type transistor (T_{W1} or T_{W0}) that presents a larger resistance.

III. HIT AND PRIORITY LOGIC AND PORT ASSIGNMENT MEMORY

In this section, we present two important components of the flexible router: the hit and priority logic and the port assignment memory. Dynamic circuitry has been used to reduce transistor count and delays. The hit and priority logic implements a priority function required to ensure a deterministic execution of the routing algorithms and select the output port that provides a short path. The port assignment memory stores the port numbers that correspond to each bit-pattern entry in the CAM.

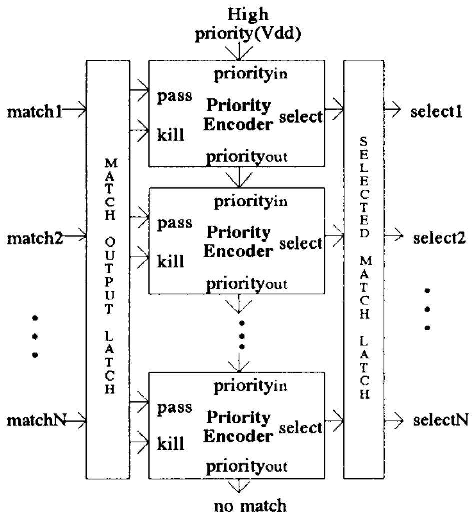
A. Hit and Priority Logic Design

The hit and priority logic (HPL) has been designed to ensure the deterministic execution of the routing algorithms. With a given input (i.e., destination address) and a set of patterns stored (program) in the matching unit, the port assignment should always be the same.

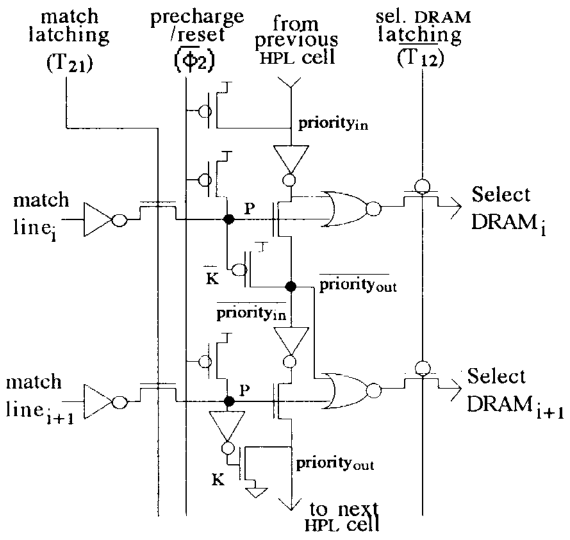
The priority function, implemented by the HPL, enables only one of the match lines from the matching unit to address the port assignment memory. In order to implement this function, we designed a priority encoder. Each cell of this encoder has three inputs [Kill (K), Pass (P), and Priority$_{\text{in}}$] and two outputs (Priority$_{\text{out}}$ and Select). Kil and Pass are generated from the match input. Table III shows the relationship between the match input (provided by the match line) and the kill and pass inputs. This table presents the values of Priority$_{\text{out}}$ as well. It should be noted that Kill and Pass are mutually exclusive inputs. When Pass is 1 (this means that there is no match at the current input), the priority value is propagated. On the other hand, when Kill is 1 (meaning that there is a match at the current input), the Priority$_{\text{out}}$ value is set to zero. Given the match input (i.e., Kill) and the Priority$_{\text{in}}$, it is possible to generate the select output, this is shown in Table IV. If there

TABLE IV
SELECT GENERATION

Match	Priority$_{\text{in}}$	Select
0	X	0
1	0	0
1	1	1



(a)



(b)

Fig. 5. Hit and priority logic design: (a) hit and priority block diagram and (b) two-cell priority encoder circuit design.

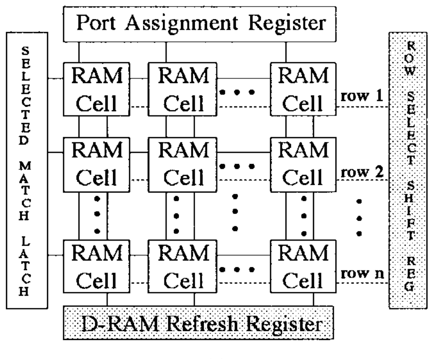
is no match the select output is always 0. When there is a match, the Priority$_{\text{in}}$ sets the select output. A block diagram of this unit is shown in Fig. 5(a).

The circuit design of two priority encoders is shown in Fig. 5(b). We have included two cells in the design of the priority encoder in order to reduce the number of buffers (inverters) in the main path of the priority chain. It should be pointed out that the number of entries to the encoder is very small (N is usually larger no larger than 16 for most of the applications). Thus, this simple circuit satisfies the requirements for this application; in addition, we have allowed close to three quarters of a cycle for this circuit to stabilize.

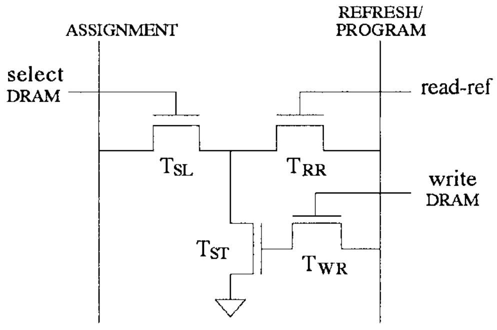
The sequence of events (which is closely linked to the timing provided in Section IV) for this unit is as follows. At ϕ_2, the priority line is precharged, and Pass and Kill are reset. The select lines (shown as Select DRAM) are set to 0 at the same time. Just after the precharge/reset time (at time T_{21}), the match condition is passed to the HPL from the matching unit. This match condition is stored in the gates of the transistor connected to this input; it should be pointed out that only gates are connected to this signal. At this time, P and K are generated, and the priority is being computed. If there is a match and the priority line is 1 (indicating that no match line above the current one is active), then the select is enabled. Thus, the priority signal that is passed to all of the cells below the current one is set to 0, preventing any further select lines from going high. Finally, the HPL output is passed to the following stage at time called T_{12}, which comes just before the next reset of the unit.

B. Dynamic Port Assignment Memory

The port assignment memory holds information about the output port that has to be assigned after a bit pattern that matches the current input is found. This memory has been implemented using a dynamic approach; Fig. 6(a) shows the organization of this memory. The selected row address is passed from the HPL. The cells in this row are read, and their data are latched in the port assignment register. This structure has a hidden refresh approach which is accomplished by means of the DRAM refresh and the row select shift registers. The hidden refresh for this memory works in a similar fashion as the one for the DCAM. Fig. 6(b) shows the circuit of a DRAM cell. In order to accommodate refresh, we have added another bus (refresh/program). It should be pointed out that the select signal is latched at the gate of transistor T_{SL}.



(a)



(b)

Fig. 6. Port assignment memory: (a) memory organization and (b) DRAM circuit design.

IV. System Timing and Router Implementation

In this implementation, we have used a two-phase clock approach. The clock diagram for match operation is shown in Fig. 7. This figure shows an instance of several consecutive input data (denoted as data1, $\cdots$). The figure also depicts how the refresh circuitry shares the BIT and NBIT bus; the rows being refreshed are CAM$_i$ and CAM$_{i+1}$. The refresh timing has been described in an earlier section; for the sake of simplicity, the refresh operation is not described any further. The destination address at the input of the SAR is latched at ϕ_1. The input data are passed on to the BIT and NBIT bus after ϕ_1. During this time, the match is being executed. Such a match is passed on to the match line at ϕ_2. Right after this clock, the match output is passed and latched at the HPL. At the end of the following ϕ_1, the HPL has completed the priority encoding; thus, the pointer to the port assignment memory is ready. The selected port assignment is latched to the port assignment register right after ϕ_1. Thus, the port assignment can be read at ϕ_2.

The timing diagram in Fig. 7 clearly shows the pipelined operation of the router. The router can sustain a throughput of one routing decision per cycle. It can be observed that the implemented router performs a single routing algorithm in 1.5 clock cycles.

The proposed router has been implemented using a low-cost 2 μm CMOS technology. The implemented router has 24 16-bit patterns (CAM size) and 24 8-bit port assignments (RAM size). This system size is large enough for all of the applications that we have studied [12]. The VLSI layout design of this router is shown in Fig. 8. Using an in-house testing board, we were able to run the system at 3 MHz. When the match line is pulled down by a single cell (within a word), it has a delay of 60 ns; it should be mentioned that this delay is measured at the output pin (there is a long delay due to the I/O pad). The propagation delay in the hit and priority logic (HPL) is 18 ns. This delay is observed when the first entry that matches the input cancels all of the other potential matches below. The pipelined router system is able to execute one route assignment per cycle. Using a submicron CMOS technology (such as 0.35 μm or smaller), the current design will be able to reach clock rates in excess of 200 MHz. The circuit was designed using scalable CMOS design rules; thus, it will be relatively easy to implement it on submicron technologies.

V. Example Application

In this section, we present a routing algorithm implementation as an application of the CAM-based router scheme. We have chosen a hypertree interconnection network [5] to illustrate the router capabilities; however, the CAM-based router can be used for a number of other interconnection networks and routing algorithms [12]. Before describing a

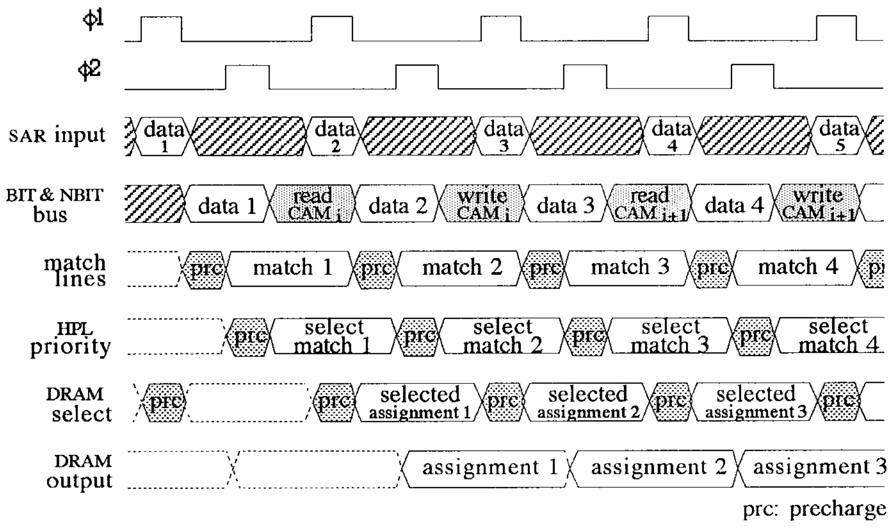


Fig. 7. Match operation mode timing.

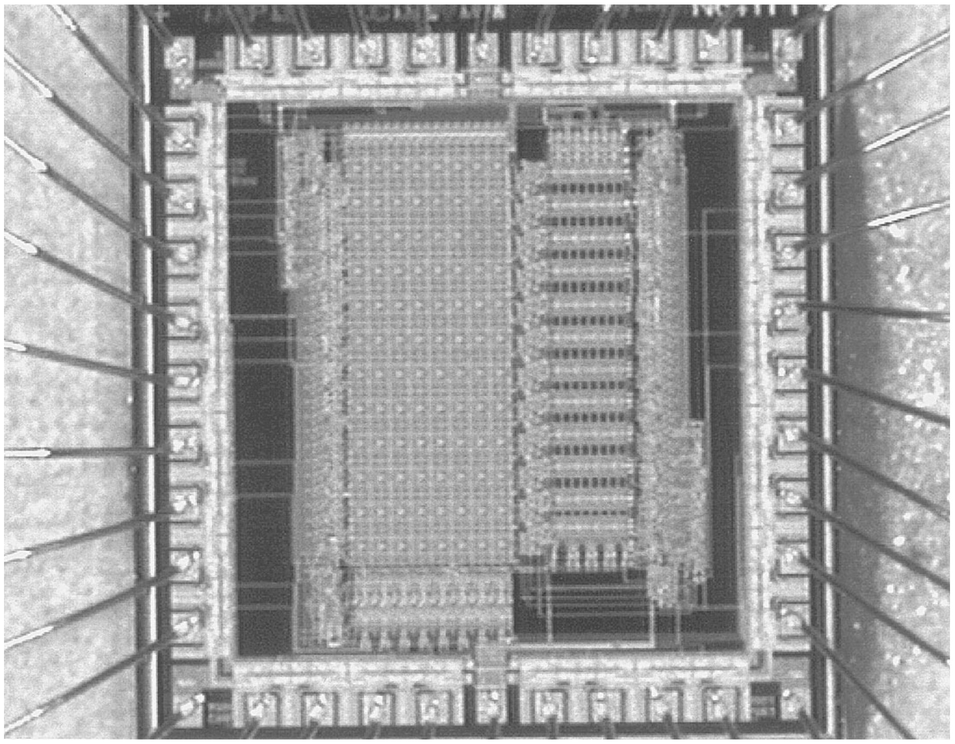


Fig. 8. Die microphotograph of the router system.

hypertree's routing algorithm, we introduce the binary tree, its addressing scheme, and its properties that are used in the algorithm. Then the hypertree, which is an enhanced binary tree, is introduced along with its features that are considered in the routing algorithm. Finally, the routing algorithm is described and its mapping onto the router's bit patterns.

A. Binary Tree Basic Structure

The binary tree structure is the fundamental building block for most of the tree networks [7]. The binary tree structure is distinguished from that of the generalized tree structures by the fact that no node has more than two children. These two children are commonly referred to as the left and right child of the node. The left (or right) child and its descendants are referred to as the left (or right) subtree of the node. Without losing generality, we have adopted the odd–even addressing scheme proposed by Horowitz and Zorat [7]. In this scheme, each node address has the format $(0, \cdots, 0, 1, c_{k-1}, \cdots, c_0)$, where the leftmost "1" bit is referred to as the leading 1 and the bits $(c_{k-1}, \cdots, c_0)$ are referred to collectively as the significant portion of the node address. The root is assigned the address of 1. As the tree is traversed downward, the address of each parent node is modified and passed on to its children. The leading 1 of the parent's address is replaced with a 0 if the address is being determined for the left child or with a 1 if it is for the right child. A new leading 1 is then appended before the significant portion of the node address.

A routing algorithm for the binary tree is described in [7]. In this algorithm, a message is first routed up the tree to a subtree containing the destination node. Once the correct subtree is reached, the algorithm sends the message down through this subtree toward its destination. We refer to these as the upward and downward phases of the algorithm. To describe a generic routing algorithm for all of the nodes of a tree, we have the following notations. The current node C has an address $(0, \cdots, 0, 1, c_{k-1}, \cdots, c_0)$, this node is at level k. The node where the message will go is called destination node D, and has an address $(0, \cdots, 0, 1, d_{n-1}, \cdots, d_0)$; this node is at level n.

A routing algorithm uses the properties of the addressing scheme. The conditions required by the algorithm include the following properties.

Property 1: The relative position of two nodes can be determined by comparing the relative position of the leading 1 in each address. The number of bits in the significant portion of a node address is equal to the level number. Thus, the position of the leading 1 indicates a node's level in the tree. For node C $(0, \cdots, 0, 1, c_{k-1}, \cdots, c_0)$ at level k, the leading 1 occurs at position k. If nodes D and C are at the same level ($k = n$), then their leading 1 must occur at the same bit position. If node D is higher (lower) in the tree than node C, then the bit position of the leading 1 in the address of D is to the right (left) of the leading 1 in the address of C.

Property 2: Nodes in a subtree inherit all of the significant bits of the subtree parent. In this addressing scheme, the child node inherits the significant portion of the parent's address. All nodes in the subtree of node C $(0, \cdots, 0, 1, c_{k-1}, \cdots, c_0)$ will have the bits $(c_{k-1}, \cdots, c_0)$ in common. Furthermore, a node in the left subtree of C will have $(c_k = 0)$ and a node in the right subtree will have $(c_k = 1)$.

B. Hypertree Interconnection Network

A hypertree interconnection topology is a binary tree network augmented with additional horizontal connections called n-cube links [5]. The hypertree structure is shown in Fig. 9. An n-cube link provides an interconnection path between two nodes in different subtrees at the same level of the tree structure. The addresses of these two nodes differ in only one bit. Using the same addressing scheme as the binary tree, we have that for a node C $(0, \cdots, 0, 1, c_{k-1}, \cdots, c_{i+1}c_ic_{i-1}, \cdots, c_0)$ at level k, the node connected by an n-cube link to node C will have address B $(0, \cdots, 0, 1, b_{k-1}, \cdots, b_{i+1}b_ib_{i-1}, \cdots, b_0)$, where $b_i \neq c_i$ and $b_j = c_j$ for all $j \neq i$. We refer to B as the hypernode of C. The bit position in which the two node addresses differ is based on the type of hypertree as well as the level in the tree at which the nodes reside. Details can be found in [5].

Having a hypercube link enhances the routing algorithm with more alternatives. The conditions required to use an n-cube link are summarized in the following properties.

Property 3: To use the hypernode, the Hamming distance between the addresses of the current node C and the destination node D must be greater than that between the addresses of B (the hypernode of C) and D. The addresses of nodes B and C differ only in bit position i. Let $d(x, y)$ represent the Hamming distance between the addresses of nodes x and y. Then $d(B, D) < d(C, D)$ requires $d_i = b_i$ or, alternatively, $d_i \neq c_i$.

Property 4: To use the hypernode, the routing algorithm must be in its upward phase. For the routing algorithm to be in its downward phase, node D must be in the subtree of node C. This requires that the bits of D and C corresponding to the significant portion of the address of C be equal. Since bit position i falls within the significant portion of the address of C, $d_i \neq c_i$ is sufficient to ensure that the algorithm is in its upward phase.

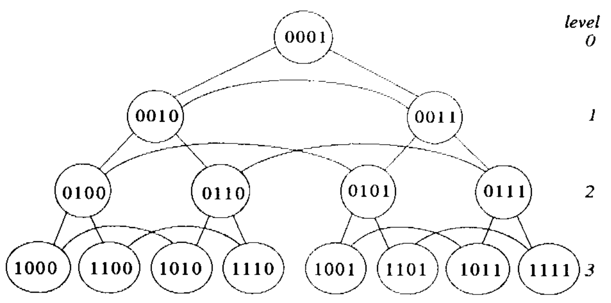


Fig. 9. Hypertree interconnection network.

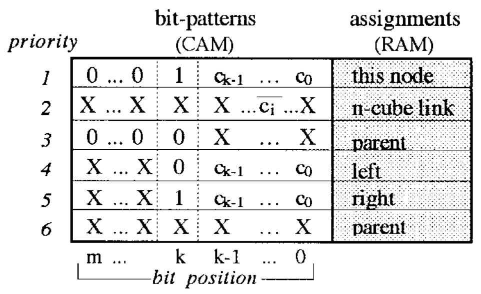
C. Hypertree Routing Algorithm and its Bit-Pattern Mapping

Based on the features and requirements of the hypertree, an algorithm has been adapted from [7]. For a message arriving at node C $(0, \cdots, 0, 1, c_{k-1}, \cdots, c_0)$ on level k destined for another node D $(0, \cdots, 0, 1, d_{n-1}, \cdots, d_0)$ at level n, a distributed routing algorithm is shown in program-like fashion at the bottom of the following page.

The algorithm consists of conditional program statements. Each of these statements is associated with a potential output port assignment (*assign*()) that could be made by the algorithm. These potential assignments are shown to the right side of the algorithm description. It should be mentioned that, once an assignment is made, the algorithm stops there, i.e., the other conditionals are not considered. We have added comments to the algorithm to make reference to what the conditional statements are checking.

To map the routing algorithm onto the bit-pattern associative router, each conditional program statement needs to be expressed in a bit pattern. This is accomplished using the four addressing scheme properties introduced in the previous subsections. The bit-pattern entries for this hypercube routing algorithm are given in Fig. 10. The entry numbers in the bit patterns correspond to the assignment number in the routing algorithm.

The first entry in the bit pattern set corresponds to the destination being the current node; thus, the destination address must exactly match the current node address. Using Properties 3 and 4, entry number 2 checks the condition to use an n-cube link; this is represented by the bit pattern $(X, \cdots, X, \bar{c}_i, X, \cdots, X)$. Entry 3 checks if the relative



priority	bit-patterns (CAM)			assignments (RAM)
1	0 ... 0	1	c_{k-1} ... c_0	this node
2	X ... X	X	X ... $\bar{c}_i$ ... X	n-cube link
3	0 ... 0	0	X ... X	parent
4	X ... X	0	c_{k-1} ... c_0	left
5	X ... X	1	c_{k-1} ... c_0	right
6	X ... X	X	X ... X	parent
bit position	m ...	k	k-1 ... 0	

Fig. 10. Bit patterns for an oblivious hypertree routing algorithm.

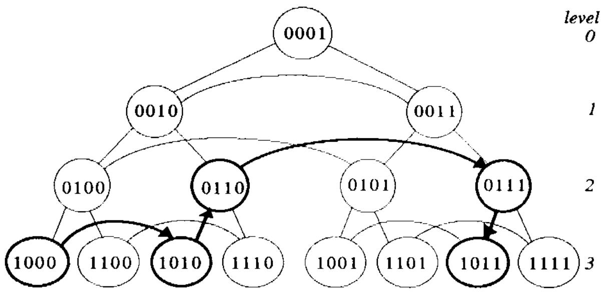


Fig. 11. Hypertree routing example.

level of the destination address is above the current node. Property 1 is used to check the level; the leading 1's for all of the nodes above the current level have zeros in bits m–k of their addresses. Using Property 2, entries 3 and 4 can be generated; all of the nodes in the current one's subtree must inherit the same bits from $k-1$ to 0. It is also necessary to check bit k of the destination node to send the message to the proper branch of the subtree. The last entry has the purpose of sending the message to the parent if none of the other conditions has been met. This entry takes care of the destination addresses that are at the same or below the current node level, but are not part of the current node's subtree.

An example of how a message will be routed in a hypertree using the bit-pattern router is shown in Fig. 11. The source and destination node addresses are 1000 and 1011, respectively. It should be pointed out that each node has its own bit-pattern associative router which has a customized set of bit patterns (these are based on the generic bit patterns shown in Fig. 10).

TABLE V
NODES AND ENTRIES USED TO ROUTE A MESSAGE FROM 1000 TO 1011

Current node	entry used	comments
1000	2	n-cube provides short path
1010	6	dest. is at same level
0110	2	n-cube link is used again
0111	4	dest. is in left subtree
1011	1	message arrives to dest.

Table V shows the nodes and entries at each node that are used to route the message from the source to the destination.

VI. Concluding Remarks

In this paper, we have presented a custom VLSI implementation of a flexible router scheme for parallel interconnection network architectures. This programmable router has been implemented using a novel dynamic CMOS content-addressable memory (DCAM) array. This array performs one comparison or match per cycle. The refreshing of the DCAM has been hidden from the match operation; the hardware resources are shared at different times during one clock cycle by these two operations. This, in turn, allows the matching and refresh operations to be intermixed.

The main features of the proposed VLSI system include the following.

- *Novel Parallel Routing Algorithm Execution*: Bit-pattern matching is used to execute the routing algorithm in parallel. All potential routing alternatives are considered in parallel. The hit and priority logic provides a fast collision resolution when multiple pattern matches are found.
- *High Performance*: The pipelined implementation provides a high-performance router with a throughput of one routing decision made at every clock cycle. A single routing decision requires 1.5 cycles to complete its execution.
- *6.5 n-Type Transistors per DCAM Cell*: This cell requires a very small number of transistors for its implementation.
- *Don't Care (X) Condition per Cell*: Each cell is capable of storing a "don't care" condition which can be extremely useful for a number of applications.

```
BEGIN
    IF  $(D = C)$    \* check if destination is the current node *\
            THEN: assign (current node)                                         Assignment 1
    IF  $(d_i \neq c_i)$    \* check if the hypernode can be used *\
            THEN: assign (n-cube link)                                          Assignment 2
    IF  $(k > n)$    \* check if destination is at a higher level *\
            THEN: assign (parent node)                                          Assignment 3
    IF  $(k < n)$    \* check if destination is at a lower level *\
            THEN: IF ($D$ in the subtree of $C$) \* check if dest. is in current subtree *\
                        THEN:   IF ($D$ in left subtree of $C$)
                                        THEN:   assign (left subtree)           Assignment 4
                                        ELSE:   assign (right subtree)          Assignment 5
                        ELSE:   assign (parent node)                            Assignment 6
END
```

- *Hidden Refresh*: The refresh for the DCAM and DRAM is accomplished in a transparent fashion. The match operation of the DCAM is not interrupted while refresh takes place.
- *Expandability*: Using the proposed approach, the DCAM and DRAM arrays can be easily expanded to accommodate the required number of bits per word and the number of words.
- *Intrinsic Pipeline Stage Latching Approach*: Data are latched in the gate capacitance of transistors used in the logic for the following pipeline stage. Thus, there is no need for extra hardware to accommodate the pipeline latch requirements.
- *Potential for Power Savings*: The DCAM cell has no direct paths between V_{DD} and ground where current could be wasted in a number of CMOS designs. In this router system, paths between V_{DD} and ground are avoided by precharging and discharging lines at different times. This, in turn, provides not only fast execution of some logic fuctions (such as match condition), but also power savings.

The prototype system has been successfully tested. The testing indicates that the system is functional and able to carry out one port assignment computation per cycle. The proposed DCAM could also be used in other applications such as data communication, video processing, and data and knowledge bases.

Acknowledgment

The authors are grateful to the anonymous referees of the paper for their helpful comments and suggestions. The authors acknowledge MOSIS for fabricating the test chips, and Binghamton University's IEEC for producing the microphotograph of the chip.

References

[1] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing*, 2nd ed. Benjamin/Cummings, 1994.

[2] Y. Chen and S. J. Upadhyaya, "Reliability, reconfiguration, and spare allocation issues in binary tree architectures based on multiple-level redundancy," *IEEE Trans. Comput.*, vol. 42, pp. 713–723, June 1993.

[3] J. G. Delgado-Frias, R. Sze, D. Summerville, and V. Aikens, "A VLSI CAM-based router for multiprocessor organizations," in *Proc. 4th Great Lakes Symp. VLSI*, Notre Dame, IN, Mar. 1994, pp. 124–129.

[4] J. Duato, S. Yalmanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. CA: IEEE Computer Society Press, 1997.

[5] J. R. Goodman and C. H. Séquin, "Hypertree: A multiprocessor interconnection topology," *IEEE Trans. Comput.*, vol. C-30, pp. 923–933, Dec. 1981.

[6] K. E. Grosspietsch, "Associative processors and memories," *IEEE Micro*, vol. 12, pp. 12–19, June 1992.

[7] E. Horowitz and A. Zorat, "The binary tree as interconnection network: Applications to multiprocessor systems and VLSI," *IEEE Trans. Comput.*, vol. 30, pp. 247–253, Apr. 1981.

[8] INMOS Ltd., *The 9000 Transputer Products Overview Manual*, INMOS Document 72 TRN 228 00, 1991.

[9] T. Leighton, "Average case analysis of greedy routing algorithms on arrays," in *2nd Annual ACM Symp. Parallel Algorithms and Architectures*, Crete, Greece, 1990, pp. 2–10.

[10] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. San Mateo, CA: Morgan Kaufmann, 1992.

[11] J. Park, S. Vassiliadis, and J. G. Delgado-Frias, "Flexible oblivious router architecture," *IBM J. Res. Develop.*, vol. 39, pp. 315–329, May 1995.

[12] D. H. Summerville, J. G. Delgado-Frias, and S. Vassiliadis, "A flexible bit-pattern associative router for interconnection networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, pp. 477–485, May 1996.

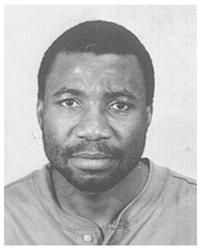
[13] ——, "A high performance pattern associative oblivious router for tree topologies," in *IPPS'94: 8th Int. Parallel Processing Symp.*, Cancun, Mexico, Apr. 1994, pp. 541–545.

[14] J. P. Wade and C. G. Sodini, "Dynamic cross-coupled bit-line content addressable memory cell for high-density arrays," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 119–121, Feb. 1987.

José G. Delgado-Frias (S'81–M'86–SM'90) received the B.S. degree from the National Autonomous University of Mexico, the M.S. degree from the National Institute for Astrophysics, Optics and Electronics, Mexico, and the Ph.D. degree from Texas A&M University, all in electrical engineering.

He is with the Electrical Engineering Department at the State University of New York at Binghamton where he is an Associate Professor. He has held academic positions at the University of Oxford, England (as a Post-Doctoral Research Fellow) and the National Autonomous University of Mexico (as an Assistant Professor). His research interests include parallel computer architecture, interconnection networks, VLSI design, computer hardware organization, neural network computing machines, and optimization using genetic algorithms. He has co-authored more than 80 technical papers and co-edited three books. He has been granted 11 U.S. patents.

Dr. Delgado-Frias has been the co-chairman of three international workshops and a program committee member of a number of international conferences. In 1994, he received the State University of New York System Chancellor's Award for Excellence in Teaching. He is a Senior Member of the IEEE and a member of the Association for Computing Machinery, American Society for Engineering Education, and Sigma Xi.



Jabulani Nyathi received the B.S.E.E. degree in 1994 from Morgan State University, MD, and the M.S.E.E. degree in 1996 from the State University of New York at Binghamton. He is currently pursuing doctoral studies in electrical engineering (computer engineering) at the State University of New York at Binghamton.

He is an Adjunct Lecturer at the State University of New York at Binghamton, where he also has served as a Teaching Assistant, Research Assistant, and Graduate Mentor for minorities enrolled in math and science. His research interests lie in the design and implementation of high speed, high density, and low power dynamic VLSI circuits.

Douglas H. Summerville (S'93–M'97) received the B.S.E.E. degree in 1991 from The Cooper Union for the Advancement of Science and Art, New York, NY and the M.S.E.E. and Ph.D. degrees from the State University of New York at Binghamton in 1994 and 1997, respectively.

He is currently an Assistant Professor in the Department of Electrical Engineering at the University of Hawaii at Manoa, Honolulu, HI. He has been a Visiting Researcher in the Mission Computers and Processors Branch at the Naval Air Warfare Center Aircraft Division, Patuxent River, MD, conducting research on parallel and distributed processing systems. His research interests include interconnection networks, parallel computer architecture and design, and VLSI design.

In 1995, Dr. Summerville received the Graduate Student Award for Excellence in Teaching. He holds memberships in the Association for Computing Machinery, American Society for Engineering Education, Eta Kappa Nu, and Sigma Xi.