J. W. Jones

Array Logic Macros

Abstract: A macro design approach is discussed which combines the cost-effective attributes of array logic structures with those of random logic. These macros utilize the following features: (a) internal feedback registers for performing sequential logic, (b) masking and submasking to reduce the number of words in the array for certain functions, (c) control of the array's output level to vary the apparent size of the array, (d) decoding on input pairs and/or EXCLUSIVE ORing on output pairs for increasing the number of logic levels, and (e) random-access memory in the feedback and its use in interrupt handling. The macros are explained by specific design examples. This paper also discusses standard logic circuits in combination with an array structure to produce a component that can be used efficiently in specific data processing areas. The designer may elect to define an array logic macro which is a combination of some of the features given in this paper. The guideline for this selection is based upon the features necessary in an array structure to be competitive with a random logic LSI chip.

Introduction

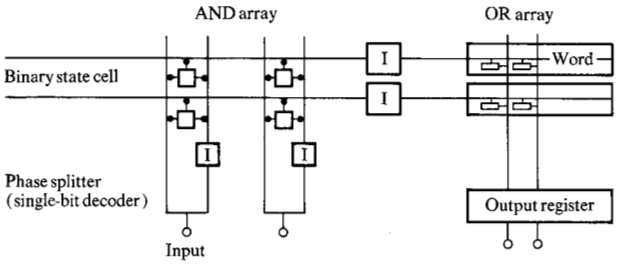
The type of array logic considered in this paper is the dual array configuration [1], where one array is a programmable decoder whose output selects words in the second array [Random Access Memory (RAM) array]. Each word in the decode array can be programmed to select its associated word in the RAM array. This selection is based upon matching the contents of the word with the input variables. The word may be programmed to match on each of the input variables being TRUE (1), INVERTED (0), or DON'T CARE (0 or 1).

The decode word is a product (AND) term since the selection is based upon specific inputs being at a defined value. The DON'T CARE condition enables the decode array to select more than one word simultaneously. The RAM array generates the sum (OR) of all the selected words on its output.

Thus, the function performed by the array logic considered in this paper is the sum of products. This type of array is programmed to solve Boolean equations which are expressed in the form of sums of products. The number of variables that can be processed in the product is limited to the number of inputs to the decode array, while the number of sum terms is restricted to the number of words in the array.

The purpose of this paper is (a) to show how this type of array logic solves specific problems in a data processing system, and (b) to demonstrate how minor modifications to this array can significantly expand the variety of its applications. The dual array structure has the same characteristics and circuits as an associative array. Throughout the remainder of this paper, the programmable decode array (which performs the product function) will be called the AND array; the RAM array (which performs the sum function) will be called the OR array.

Figure 1 The binary associative array.



Features of array logic

This section describes the features, operation, and attributes of array logic. The next section discusses additions to array logic for specific applications.

• *Associative array*

An associative array is used for table lookup, and is analogous to the telephone directory. Just as the telephone directory contains the names of the subscribers and does not contain the exhaustive combination of all possible names, so the associative array translates certain (but not all) combinations of inputs into outputs.

The input to the associative array is a binary code, and a word in the array is said to match the input when its binary pattern is the same as the pattern in the input register. The word that matches the input reads out the contents of its associated word in the OR array. The associative array performs a translation function. Thus, the telephone directory analogy demonstrates the advantage of an associative array over a conventional array, since a conventional array would have to maintain the exhaustive combination of all possible names in the alphabet. (Table compression techniques are not considered in this comparison.)

• *Array logic structure*

An implementation of the associative array is shown in Fig. 1. The phase splitter on the input to the array enables each word to select either the true (1) or invert (0) value of the input. The binary state cell makes this selection at the node between the rows and columns (Fig. 2). This unidirectional cell is personalized by removing one of the devices during a mask step or by laser technique. The truth table for the binary associative cell is a match when the input and the cell state are the same, a mismatch when they are different.

• *The associative function*

A word is selected in the associative array when the states of all the cells in that word match the corresponding input binary bit. The associative array is called the AND array because the matching process in the word of an associative array is equivalent to a logical AND function, whereby the inputs to the AND are defined by the state of each cell, i.e., true (A), invert ($\bar{A}$), or DON'T CARE (independent of variable A and both devices removed in the binary state cell shown in Fig. 2).

The AND operation is performed in the array by the use of DeMorgan's theorem ($A \cdot B \cdot \bar{C} = \bar{A} \vee \bar{B} \vee C$). A word K in the AND array has N cells and is selected when Eq. (1) is satisfied. C_{KP} is the cell in word K and related to input I as position P.

$$(C_{K1} = I_1) \cdot (C_{K2} = I_2) \cdot (C_{K3} = I_3) \cdot \text{- - - -} \cdot (C_{KN} = I_N) = \text{Match.} \quad (1)$$

Applying DeMorgan's theorem, we obtain

$$\overline{(C_{K1} \neq I_1) \vee (C_{K2} \neq I_2) \vee (C_{K3} \neq I_3) \vee \text{- - -} (C_{KN} \neq I_N)} = \text{Match.} \quad (2)$$

The inversion of the total expression (2) is achieved by the inverting circuit between the AND and OR arrays (see Fig. 1). The OR operation ($\vee$) in Eq. (2) is performed on the Word Select line by all the cells in that word. The nonequivalence in Eq. (2) is achieved by the personality or state of the cell.

• *Programming the* AND *array*

A simple method for programming the AND array is to write a 1 or 0 in the word corresponding with the input variables, depending upon whether the word selected on the input is TRUE or INVERTed, respectively. Blanks can be used to represent the DON'T CARE state. This code is readable when arranged in rows as in the array. Experimental software programs (in APL) have been written to operate on arrays coded in this manner so as to generate the information for the chip personalizing mask.

The three coded states can be easily transformed into the cell states. A match on the TRUE (1) input implies a mismatch when the input is the INVERSE (0). This matching requires a device (e.g., a diode) on the column which has the inverted input (see Fig. 2) and no device on the direct column. A match on the INVERTed (0) input implies a mismatch when the input is TRUE (1). This is realized by having a device on the direct columns and no device in the column which has the inverter (see Fig. 2).

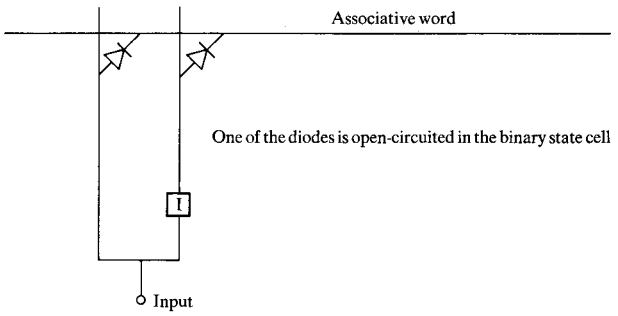
The DON'T CARE state is a state of the associative array cell which enables the word to be matched with either 0 or 1 input to that column. It can be seen from Fig. 2 that if both diodes are open-circuited, then the matching of the associative word is independent of the input. The fourth state (cell state 11) has no practical function since a match cannot be obtained (Table 1).

The DON'T CARE state enables multiple words to be selected simultaneously in the associative array. The output from the OR array is the OR of all the selected words.

Table 1 Truth table for the four-state cell.

	Cell state			
Input	*00*	*01*	*10*	*11*
0	Match	Mismatch	Match	Mismatch
1	Match	Match	Mismatch	Mismatch

Figure 2 The binary state read-only cell.



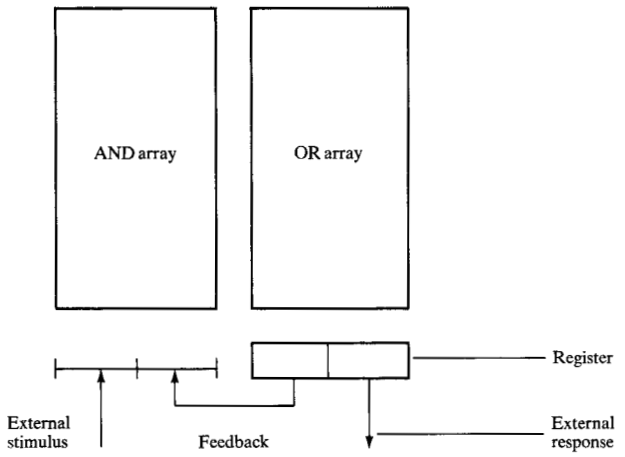


Figure 3 Sequential switching circuit.

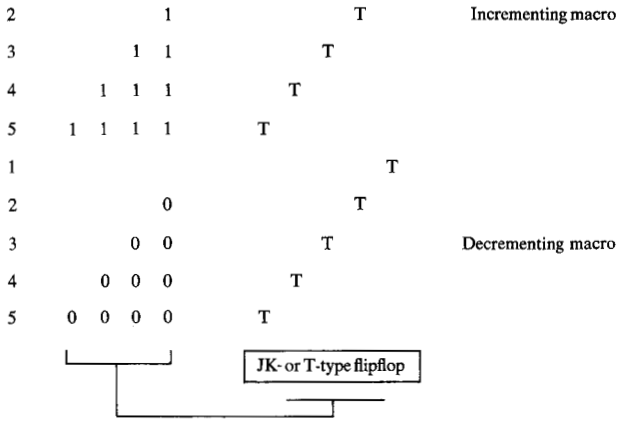


Figure 4 Incrementing and decrementing macros.

• *Programming the* OR *array*

The OR array is programmed by writing a 1 in the position where an output is required when that word is selected. The OR array consists of a binary state device, i.e., it generates a 0 or a 1 output when selected. A 0 can be defined by a blank during the programming of the OR array. An exception to this binary output coding is when the OR array operates on JK-type flipflops or RS-type flipflops on the output. In this case, symbols like S-R-T are necessary to indicate the operation performed by the flipflop registers: S is the symbol for the setting of the register, R is the reset, and T is the toggle operator. The generation of the mask personalization for the J, K, R, or S input to the flipflop register from the above code can be done manually or by a software program.

• *Combinational logic*

The array solves logical functions which are expressed as sums of products; hence, the designer must convert his expressions into this form. Furthermore, the efficiency of the array is improved if the number of sum terms is minimized so as to use the least number of words in the array. (Procedures such as the Quine-McCluskey method are useful for such minimization.)

One of the main advantages of array logic is that a library of tested codes which perform specific functions can be built up and used as a design aid. These functions, which are expressed as sums of products, are defined as array logic macros. This is a significant advantage for array logic because the interconnection of macros in random logic can present severe topological problems. Race conditions are avoided in array logic by providing a clocked register on the output of the OR array. The status of this register is changed by the output from the OR array at a specific clocked period, and the rate of the clock is dependent upon the propagation time through the array.

Additions to array logic for specific applications

This section considers modifications to the simple array logic structure so as to expand its range of application. The modifications are in the form of random logic circuits which are combined with a single AND/OR array. This composite combines the ease of design and fabrication that is characteristic of array logic with the efficient use of silicon typical of random logic.

• *Sequential logic*

Sequential logic consists of transforming the internal state (internal feedback register) and/or generating external outputs according to the current state and the external input.

The register on the output of the OR array may be either D-, JK-, RS-, or T-type flipflops or a combination of these types. In general, the delay (D-type) flipflop is used where the information is set during each cycle, whereas the JK-, RS- or T-types are used for sequential switching purposes. The JK- and T-type flipflops are particularly suited for the incrementing and decrementing macros. The outputs of the JK- or T-type flipflops are fed back to the AND array to form the memory part of a sequential switching circuit (see Fig. 3).

Incrementing and decrementing macros

The incrementing and decrementing macros for a five-bit counter are shown in Fig. 4. The T symbol in the OR array indicates the toggle operator on the JK- or T-type flipflops.

The operation of the incrementer can be explained by defining the algorithm as a toggle (complement) operation on all the propagate terms (leading 1 bits) of the count including the sink (0 bit). Note that the lowest value of the count is always toggled (word 1), and the next higher position is toggled only when the lowest bit is 1 (word 2). The operation of the other words in the

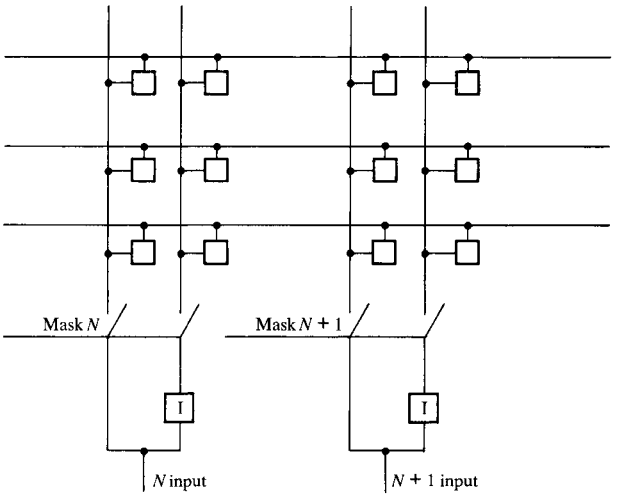


Figure 5 Masking in the AND array.

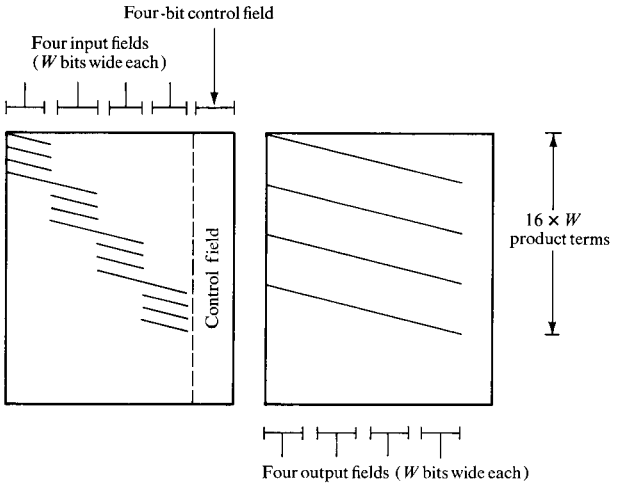


Figure 6 Data path switching application.

increment table are self explanatory. It can be seen that incrementing a count value of three selects words 1, 2, and 3 in the incrementing macro.

The decrementing and incrementing macros have similar formats, the difference being that the decrementing macro uses the propagation of the borrow instead of the carry. (Note that word 1 is shared by both macros.)

Finite state switching

The configuration of the AND and OR arrays which have the RS- or JK-type flipflop register in the feedback (Fig. 3) is a sequential switching element. The current state which is maintained in the feedback is combined with the external stimulus in the AND array to select words in the OR array. The selected words generate operations which modify the contents of the feedback register so as to produce the next state. The state switching algorithm is easily expressed in the form of a sum of products; however, the encoding of the states should exploit the DON'T CARE facility in order to minimize the number of words necessary to define the state transition.

One method of minimizing the state-switching algorithm is to divide the total number of states into subsets according to common external switching inputs. These subsets are coded such that each subset can be defined uniquely by the minimum number of words. Many terms in the subset are combined into a single word by the use of the DON'T CARE facility. These words satisfy the switching conditions necessary to transfer all of the states in the subset into a new state. The external inputs should also be encoded into groups that can be expressed by a single product term by the use of the DON'T CARE condition. The DON'T CARE facility is significant in compressing the state transition algorithm. The procedures for selecting and coding subsets, and for coding the external input, are part of the state-assignment phase of the design.

• *Masking operation and its application to data path control*

The masking function in associative array logic has been described [2] as a means of degating the input variables to the AND array (Fig. 5). The masked columns in the AND array are forced into the DON'T CARE state. The truth table of the mask operation is given in Table 2.

Masking in the AND array is important in the control of data paths, whereby the array becomes an efficient crosspoint switch. Figure 6 shows four input fields (each field W bits wide) and four output fields (each field W bits wide). Each of the four input fields may be switched to any of the four output fields; hence, there are 16 possible combinations of data paths. The number of product terms necessary to perform this function is 16 $\times$ W. The particular data path is selected by a four-bit-wide control field.

Table 2 Truth table of mask operation.

		Cell state		
Input	*Mask*	*0*	*1*	*Don't care*
1	0	*	–	–
0	0	–	*	–
1	1	–	–	–
0	1	–	–	–

– Match condition
*Mismatch condition

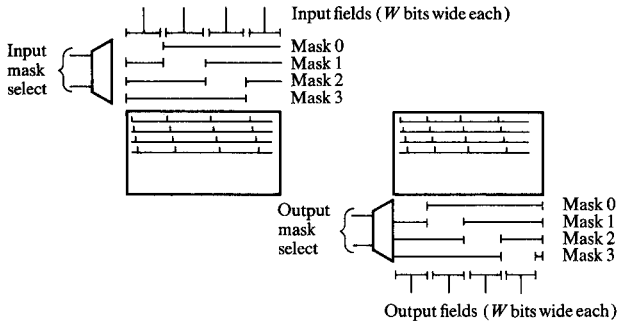


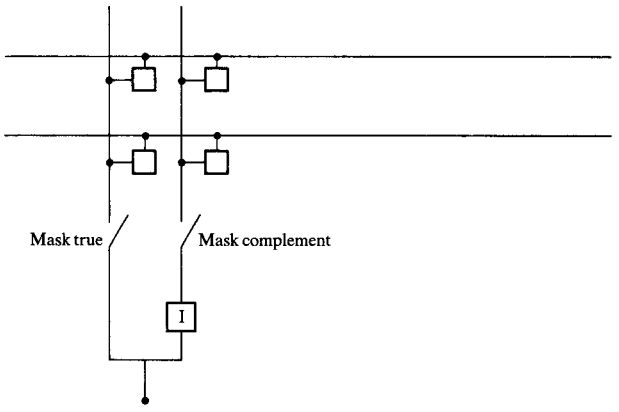
Figure 7 Data path switching application with masking on input and output.

The advantage of the mask operator is demonstrated in Fig. 7, where the same function is performed with only W product terms. The four-bit control field is divided into two pairs of control bits. The first pair selects one of four masks for the input to the AND array; the second pair selects one of four masks for the output of the OR array. Each of the four masks on the input provides a mask operator over three of the four input fields, thereby reconfiguring their associated cells to DON'T CARE. Similarly, each of the four masks on the output of the OR array inhibits output on three of the output fields. A factor of 16 reduction was shown in this application of masking; however, this ratio varies according to the number of input and output fields. The masking operation can be defined so as to enable multiple fan-outs; i.e., any single input to the AND array can be switched to any number of outputs by the selection of mask fields.

• *Submasking and its application to the overlaying of macros*

Submasking can be defined as the ability to reconfigure a cell into states other than DON'T CARE; this result is achieved by providing separate controls to each of the outputs of the decoder (Fig. 8). Submasking enables certain sum-of-product terms to be overlaid in the same words in the array. The merged terms are selected by the use of submasking, which configures the cell into the required state. An example of the application of submasking is the provision of the transfer of either the true or the complement value of a variable between the input and output by the use of the same product terms. A cell which is written in the FALSE state (11) is configured to either 01 or 10 by masking the true or complement output of the decoder. Masking the true (direct) output from the decoder configures the FALSE state cell to the 1 state, whereas masking the complement configures it to the 0 state (i.e., generates the complement function in the above example).

Figure 8 Separate controls for true and complement of input.



The increment and decrement macros (Fig. 4) are identical tables and can be superimposed by using the submasking feature.

• *Control of output and its application to variable array geometry*

Control of the output signal levels (TRUE or INVERSE) enables a change in the apparent size of the AND/OR array. This change can be made in both width (number of product terms) and depth (number of sums of products) of this array.

It is necessary to minimize the number of variables into the AND array in order to keep the cell utilization high, because a large number of input variables into the AND array necessitates a large number of DON'T CARE states. The number of equivalent circuits of the array is increased by having the ability to do the AND function on the bus that interconnects the arrays. This increase is achieved by providing the complement of the output variables, and thereby performing the dot-AND function on the external bus. The dot-AND is realized by using DeMorgan's theorem (e.g., $A \cdot B \cdot \overline{C} = \overline{A} \vee \overline{B} \vee C$). The outputs are designed so as to provide the inverted variables $\overline{A}$, $\overline{B}$, and C. The OR is performed on the bus while the overall INVERT is coded in the AND array at the destination.

The apparent number of words can be increased by performing the sum function on the bus that interconnects the arrays. This increase is achieved by the use of the dot-OR function on the bus.

• *Increasing the number of logic levels in the* AND/OR *array*

The two levels of logic provided by the AND/OR array can be supplemented by logic circuits so as to increase the number of logic levels in the chips. This increased level of logic improves the applicability of the array.

Two-bit decode

One method of increasing the number of logic levels is to perform partial products on the input variables prior to the AND array. The provision of a two-bit decode on pairs of the input variables is one method of achieving two extra levels of logic. The two-bit decode provides the same number of minterms to the AND array as a single bit decode; that is, $A \cdot B$, $\overline{A} \cdot B$, $A \cdot \overline{B}$, $\overline{A} \cdot \overline{B}$ are the output combinations obtained from the two-bit decoder, while A, $\overline{A}$, B, and $\overline{B}$ are the outputs produced by two single-bit decoders.

The partial product of a large number of variables can be performed externally to the chip by the dot-AND feature discussed in the previous section. This technique eliminates the wasteful use of input pins to the array, and also increases the array utilization by not wasting columns for the unused minterms in the AND array.

EXCLUSIVE OR *on the output and its application to the* AND *operation*

Another method of increasing the number of logic levels in the AND/OR array is to divide the OR array into two arrays (P and Q). Outputs from both sections of the OR array are connected to an EXCLUSIVE OR circuit, and the result is connected to the output (Fig. 9). The EXCLUSIVE OR, or pair of outputs, provides the mechanism for the masking function on the output.

Applications that take advantage of the function provided by the EXCLUSIVE OR on the array output include the addition and subtraction operations. The addition of two arguments is given by $A \forall B \forall C$ ($\forall$ is the symbol for the EXCLUSIVE OR operation), where A and B are the two arguments and C is the carry term. $A \forall B$ can be expressed in sum-of-product terms; this sum is coded into the P array. Similarly, the carry C can be expressed as a sum of products; this sum is coded into the Q array. This configuration of standard logic circuits combined with the AND/OR array results in a component which does the addition operation in a single array cycle.

- *Interrupt handling or event processing*

It has been shown that the AND/OR array combined with a feedback path via a register can perform sequential logic operations. The inclusion of a random access memory in the feedback path provides extra storage function. This storage function can be used to store data or control information, which gives it the ability to do multiple sequential logic operations in parallel. An enabled interrupt request can be detected in the AND array; this request initiates the storing of the current status at a fixed location ($D1$) in the random access memory. The procedure continues by reading out the routine address from another fixed location ($D2$) which is associated with the specific interrupt. The interrupt processing terminates by invoking a macro which reads out the saved word from the location ($D1$), and the dumped program continues. The number of levels of interrupts that can be processed is determined by the number of words available in the random access memory.

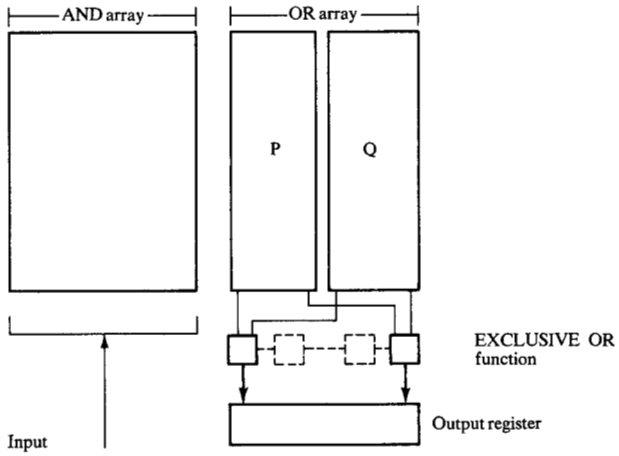


Figure 9 The EXCLUSIVE OR function on the array output.

Nested interrupts are also possible with this configuration. A sequence of macros may write the entry address of another sequence in a word $R1$ in the random access memory and save its own address at location $R2$. The branching to the macro sequence addressed by $R1$ is achieved by reading its contents, while the return address is restored by invoking a macro which reads the adjacent address $R2$. Routines may be nested until all allocated words in the stack have been used.

Summary

This paper has described the features that can be combined with the basic AND/OR array so as to significantly improve its efficiency without detracting from its design and manufacturing advantages. These features include:

- *Internal feedback register*, which provides sequential logic capability,
- *Masking and submasking*, which provide the ability to combine several macros into the same words,
- *Two-bit decode and the* EXCLUSIVE OR *function*, either of which provides extra logic levels,
- *The random access memory*, within the array feedback, which provides multiple sequential processing capability.

At the present time, array logic in the form of the AND/OR array (with or without a feedback register) is available for use by the system designer. As a matter of fact, several small systems that utilize these arrays are already on the market. Because of the ease of programming and debugging the AND/OR array logic and the

simplicity with which the array logic chips are produced, this macro design approach represents a relatively low cost entry into LSI.

Acknowledgments

The author thanks J. C. Logue and the staff of his IBM Fellow Program at the System Products Division Laboratory in Poughkeepsie, New York, for their assistance.

References

1. J. W. Jones, J. F. Sears, and K. G. Taylor, "Associative Store," Patent No. 2168409 (France).
2. B. T. McKeever, "The Associative Memory Structure," *AFIPS Conf. Proc., Fall Jt. Comput. Conf.* **27,** Part 1, 371 (1965).

Received April 3, 1974

The author is located at the IBM United Kingdom Laboratories, Hursley, Winchester, Hampshire, England.