INTRODUCTION

The cobweb cellular arrays are embellishments of the cutpoint cellular array that are made by complicating the cell-interconnection structure. This new class of arrays will allow for more economical and efficient logical designs than are possible in cutpoint arrays. As a background to the new arrays, the properties of the cutpoint array will be reviewed.

CUTPOINT ARRAY

The cutpoint cellular array is a two-dimensional rectangular arrangement of cells, as shown in Fig. 1b, each cell has binary inputs from neighboring cells on the top and the left, and binary outputs to neighboring cells on the bottom and right. In addition to being used in the cell, the input to the left of each cell is bussed to the right output. The bottom output of each cell is set as one of six combinational switching functions of the two cell inputs, or as an R-S flip-flop—in either case by the use of four specification bits, or cutpoints, in each cell. By specifying these cutpoints independently for every cell of a cutpoint cellular array, the array is thereby particularized to a required logical property. The table in Fig. 1a includes the logical functions that can be produced at the bottom output of each cell, depending on the particular specification of its cutpoints. The symbol F in the index column indicates an R-S flip-flop.

A 3 × 4 array of cutpoint cells is shown in Fig. 1b; the specification bits are indicated as dots. A realization for one cutpoint cell in terms of diode-transistor circuits is shown in Fig. 2. The four cutpoints in this realization are depicted as switches; however, they could be photoresistors, flip-flops, or breaks or bridges in conductors. The DTL realization in Fig. 2 is one of many circuit possibilities for a cutpoint cell.

PROBLEMS WITH CUTPOINT ARRAYS

It has been shown in the previously cited references that arbitrary logical functions can be realized using appropriately specialized cutpoint arrays. However, certain of these realizations tend to be inefficient in terms of the number of required cells. For instance, the best-known realization for a three-bit parallel adder using no more than two cutpoint arrays is shown as Fig. 3. In this figure the two three-bit words (a3, a2, a1) and (b3, b2, b1) are added to form...
the sum word \((s_3, s_2, s_1)\). The input carry to the low-order column is \(c_0\), while \(c_1\) is the overflow bit. An \(n\)-bit parallel adder can be formed in a similar manner to the one in Fig. 3; a total of \((2n + 1)^2\) cells are required in two adjoining arrays for such a realization.

Reference back to Fig. 1a shows that cells with an index 1 form the complement of the top input. This one-variable function is convenient to use when transmittal of information vertically in an array is desired. Hence, vertical cascades of Index 1 cells in a cutpoint array indicate that in effect, no logic is being performed, perhaps with the exception of one cell in each such cascade. With this in mind, it is now observed that in the upper 3 \(\times\) 7 array of Fig. 3 only six cells, roughly along the diagonal from the upper-right to lower-left corners, are used logically. Similarly, in the lower 4 \(\times\) 7 array in Fig. 3, only cells in the upper-right triangular area are used logically.

A wastage of cells similar to that encountered in Fig. 3 had been observed in several cutpoint-array logical designs, particularly in designs which involve parallel operations. This inefficient use of cells occurs in most of these situations because every bit of one operand word must interact with every bit of a second operand word. In a cutpoint array the only convenient way this interaction can occur is to introduce the bits of one word on the side of the array and the bits of the other word along the top. This orthogonal introduction of the two operand words into a cutpoint array seems necessary because no facility is provided within the array to change the direction of information flow from vertical to horizontal.

It it were possible to redirect the information
Figure 2. Circuit for one Cutpoint Cell.

Figure 3. Cutpoint realization for a three-bit parallel adder.
flow inside a cellular array, two n-bit operand words might be applied to the side of the array, (or both to the top), and possibly a significant reduction would result in the number of cells that are required. Instead of requiring \(0(n^2)\) cells, the resulting array might require \(O(n)\) cells. Thus this lack of control on the direction of information flow constitutes an important problem in the use of cutpoint arrays.

Another problem encountered in practical logical designs using cutpoint arrays is the excessive requirement for jumper connections from edge-output points to edge-input points of the same array. One example of this problem is shown in Fig. 4, which is a five-bit shift register driven by a four-phase clock. For this example four such jumpers are used. Jumpers of this type will be termed *edge jumers*. A second example is shown in Fig. 5. In this fig-

\[
\begin{align*}
E_{66} &= \Sigma(1, 6) \\
E_{43} &= \Sigma(0, 1, 3, 5) \\
E_{129} &= \Sigma(0, 7).
\end{align*}
\]

![Figure 4. Cutpoint realization for a five-bit shift register.](image)

![Figure 5. Cutpoint realization for three functions of three variables.](image)
A requirement for edge-jumper connections in a cutpoint array often carries with it a wastage of cells. In the bottom half of the array in Fig. 5, for example, only three cells are used other than for transmitting signals.

A third problem often encountered when practical logical designs are made in terms of cutpoint arrays is an insufficient number of edge connections to the array. A final problem is the desirability to have the cells isolated from one another during the early part of the production so that it is possible to identify faulty cells by step-and-repeat testing.

The cobweb array is proposed as a means of meeting all of these problems of inefficiency for parallel operations, excessive edge-jumping, insufficient edge connections, and lack of cell isolation.

**COBWEB ARRAY**

A 4 \times 4 cobweb array is shown in Fig. 6. It is seen that within the array each cell has five possible inputs: two from a horizontal and vertical bus and three from nearby cells. Connections from edge cells to the package terminals are shown on Fig. 6 by peripheral dots. For terminal connections, each cell on the left and bottom edges of the array has one non-bus output connected to terminals, each cell on the right and top edges of the array has one non-bus input connected to terminals, and each horizontal and vertical bus is connected to a terminal. For an M \times N-cell cobweb array it is easily seen that 3(M + N) - 2 package terminals are needed. This compares with M + 2N terminals for the cutpoint array of the same size; for square arrays approximately twice the number of terminals are provided by the cobweb array, while in general the number of terminals in the cobweb array varies from one and one-half to three times the number in cutpoint arrays of the same dimensions.

In Fig. 7a one internal cell of this cobweb array is shown with its five inputs labelled as u, v, w, x, and y. Fig. 7b shows how this cobweb cell can be fabricated from the previous cutpoint cell and fourteen additional cutpoints. Of course, as mentioned previously, technologies other than the diode-transistor method shown in Fig. 7b can be used. The added cutpoints are labelled e, f, ..., r. In order to have all single-throw cutpoints, the double-throw cutpoint a in Fig. 2 is replaced in Fig. 7b by two cutpoints, g and e, where a = g'e and a' = ge'.

It is anticipated that cobweb cellular arrays will be made by one of the modern batch-fabrication technologies, such as that of integrated circuits. In making these arrays with integrated circuits, the number of deposition steps is of related economic interest. Returning to Fig. 6, and using the nomen-
TRANSISTORS: 2N706
DIODES: IN4009

Figure 7. Diode-transistor realization of cobweb cell.

clature of Fig. 7 a, it is seen that if the w busses are moved to the right of the center in each cell, all w, v and y interconnections may be deposited simultaneously. After depositing an appropriate insulating layer, the u and x interconnections together with connections for power and ground may be formed as a second deposition layer. Hence the interconnection structure of the cobweb array in Fig. 6 is two-layered. Similar reasoning applied to Fig. 1b shows that the interconnection structure of the cutpoint array is single layered.

In summary, the cobweb array consists of cells that have the same amount of electronics as cutpoint cells. Each cell in the new array has about four times the number of cutpoints as the cutpoint cell, one and one-half to three times the number of package terminals as a cutpoint array of the same size, and a two-layered rather than a one-layered interconnection structure. It will now be shown that the use of this more complicated cellular array at least partially alleviates the previously-discussed problems of cutpoint cellular arrays.

LOGICAL DESIGN WITH COBWEB CELLULAR ARRAYS

In cutpoint arrays, switching functions are produced by forming one or more vertical cascades of cells. In the cobweb cellular arrays, these cascades of cells no longer are required to be vertical. Indeed, a cascade in a cobweb array may be any chain of cells that follows the arrowheads in Fig. 6. This property of cobweb arrays gives the logical designer a considerable degree of flexibility in forming his design. The need for an increased ratio of edge connections to cells is met in cobweb arrays. By introducing other assumptions on edge connections it is possible further to increase this ratio if additional logical design experience shows this to be desirable.
In the cobweb array it is possible to use some of the cutpoints in a cell in lieu of edge jumpers. For instance, if cutpoints h and k (Fig. 7 b) are closed, this causes the x bus and input u to that cell to be connected together. Similarly, by closing cutpoints j and f, and by opening cutpoint r, the cell output can be jumpered to the w input bus; for this connection, the logical function produced by the cell is immaterial. It is also possible to jumper as many as all five inputs and the output of a cell together. Indeed, for those cases where sneak paths are not introduced, it is possible to form one jumper path among the cutpoints f, h, i, j, k, l and a second one among m, n, o, p, q. Cells that are specialized in this way are called \textit{jumper cells}. The jumper connections are designated by circling the inputs (and output) that are jumpered together and by inserting the symbol J inside the cell. If two isolated jumpers are used, triangles will designate the inputs (and output) on the second jumper.

It should be observed that J cells in the cobweb array are logically inactive. That is, jumper cells are used only to make local connections in the array, and not to perform logical operations. It should also be noted that jumper-cell connections can be made in such a way as to allow information flow in violation of the arrowheads in Fig. 6.

It is also possible to use the cutpoints h, i, ..., q to obtain an OR of two or more of the five inputs to a cell. For instance if cutpoints k, l, m, n and o are closed, then the horizontal input to the cell is \( x + y \), while the vertical input is \( u + v + w \). Care must be taken when using this property to avoid the introduction of sneak paths.

Two decomposition methods will now be shown in order to illustrate the elimination or reduction in the use of edge jumpers by the jumper-cell specialization. It is supposed that a switching function \( E = E(x_1, x_2, \ldots, x_n) \) is not producible in one cascade of cells. This function can be decomposed on one of its variables, \( x_i \), in several ways, including a form due to Shannon,

\[
E = G x_1 + H x_i' 
\]

and a form due to Reed,

\[
E = A x_i \theta B 
\]

where \( \theta \) is the EXCLUSIVE-OR operator, and each of \( G, H, A, B \) is a switching function of no more than \( n-1 \) variables, and is independent of \( x_i \).

If it is assumed that \( G', H', A', \) and \( B \) are each producible in one cascade of cells, the cobweb arrays of Figs. 8a and 8b correspond to the two decompositions of Eqs. (2) and (3), respectively. If one or more of the four \((n - 1)\)-variable functions are not producible in a single cascade, either of the above decompositions may be repeatedly applied until all subsidiary functions are realizable in one cascade.

The J cell in Fig. 8a with the y input and the z output circled means that the switches f and 1 (see Fig. 7) are closed and that switch r is open. This connects the y input to the cell output without the use of an external jumper. For cobweb cells that produce one of the functions listed in Fig. 1a, a function index is placed inside the cell and the particular inputs (if any) that are connected through the cutpoints h, i, ..., l, (Fig. 7), are designated by \textit{circles}, while the particular inputs (if any) that are connected through the cutpoints m, n, ..., q are designated by \textit{triangles}. For instance, the cell with index 5 in Fig. 8a means \((b, c, d) = (1, 0, 1)\), and since a circle is on the x bus and a triangle is on the y input, then \((h, i, j, k, l) = (0, 0, 0, 1, 0)\), and \((m, n, o, p, q) = (0, 0, 0, 0, 1)\), and finally \((e, f, g, r) = (0, 0, 1, 1)\).

As shown in Fig. 8, no jumpers are needed in the cobweb-array realization of either decomposition. Furthermore, it is noted that only one row of cells is needed for each application of the Reed decomposition.

With none of the cutpoints h, i, ..., q closed, all cells are isolated in the cobweb arrays. Therefore, step-and-repeat testing is possible for cobweb arrays that are fabricated as monolithic integrated circuits, while it is not possible for the originally proposed cutpoint arrays.

The particular interconnection structure of the cobweb array was chosen for several reasons. As the number of potential inputs to each cell is increased, the number of interconnection possibilities also increases. But this increase is obtained at the cost of additional cutpoints in each cell. Hence, it is desirable to introduce only as much interconnection versatility as the typical logical designer would use.

\footnote{In order to simplify the artwork, the terminal conventions adopted in connection with the discussion of Fig. 6 will not be explicitly shown on this and on following cobweb-array designs.}
Figure 8. Shannon and reed decompositions using cob-web arrays.
Referring back to Fig. 7a, the x and y inputs are carried over directly from the previous cutpoint array. The u input allows the designer to build up a carry propagation chain within a horizontal row of register cells. The vertical bus allows one to jumper a bottom-cell output of an array to a top-cell input. Finally, the v input is a knight’s move away so that it is possible to build up a cutpoint cascade that crosses other such cascades. The desirability of having crossings in cellular arrays has been observed before.*

A number of obvious variations of the cobweb array is possible. For instance input v, or both inputs v and w in Fig. 7a may be omitted in each cell, with a corresponding saving in cutpoints. In the latter variation, a single-layered interconnection structure results. Similarly, it is possible to invent more complicated variations of the cobweb array.

Illustrations of logical designs using cobweb arrays are given as Figs. 9, 10 and 11. These figures should be compared directly with Figs. 3, 4, and 5, respectively. First comparing Figs. 3 and 9, it is seen that an n-bit parallel adder can be synthesized using 9n + 3 cobweb cells in a single array, while (2n + 1)^2 cells in two adjoining arrays are required if cutpoint cellular logic is used. Thus, for example, a 30-bit parallel adder requires 453 cells in a cobweb array.

*By Marvin E. Brooking, private communication.
A comparison of Figs. 4 and 10 shows that all edge jumpers are eliminated in the cobweb realization of a shift register at the cost of one extra row of cells. Finally, comparing Figs. 5 and 11, it is seen that the three edge jumpers as well as half the total number of cells are eliminated when a cobweb array is substituted for a cutpoint array.

FAULT AVOIDANCE METHODS

In regard to the cutpoint cellular array, methods have been demonstrated for replacing faulty cells with spare cells. These methods no longer are feasible with the cobweb arrays; therefore, it is necessary now to develop an alternative faculty cell avoidance algorithm. It will be assumed that the faults are "electronic;" that is, a transistor has a low beta, or it has an emitter-collector short, or a diode is open-circuited, etc. All conductors and cutpoints will be considered perfect, and furthermore, the circuit design is assumed to be such that no failure condition will cause the shorting or opening of a conductor or the shorting of a power supply. It appears from a consideration of the integrated circuit technology that these assumptions are realistic.

Two clusters of cells called supercells are defined by Fig. 12. The shaded cell in each $2 \times 2$ cobweb array has five inputs (marked with the symbol $I$) that are geometrically equivalent to the cobweb cell of Fig. 7. The jumpers between points $p$ and in Fig. 12 are used for transmitting the knight's move interconnections. The supercells of Fig. 12 are arranged in such a way that one may first perform a logical design in terms of a conventional cobweb array, and then replace each cell in the first and all odd-numbered rows with a type $\alpha$ supercell. The cells in the second and all even-numbered rows are replaced with a type $\beta$ supercell.

The effect so far has been to increase the number of cells in the cobweb array by a factor of four. In this supercell array it is possible under most conditions to make focal perturbations of the logical de-

![Figure 11. Cobweb realization for three functions of three variables.](image)

![Figure 12. Cobweb supercells.](image)
Figure 13. Exhaustive listing of the cobweb array fault-avoidance algorithm.
sign in order to avoid faulty cells. Assuming that only one or two of the five inputs to a given cell are connected by means of cutpoints h, i, . . . , q in Fig. 7, it is necessary to demonstrate a fault-avoidance algorithm for $C_5^2 = 10$ cases (the two-input cases cover the one-input cases). However, the logical cells in a supercell array appear in two geometrically different environments that correspond to the types $\alpha$ and $\beta$ supercells; therefore, a total of 20 cases must be investigated. Proceeding by exhaustion, a fault-avoidance algorithm for each of these 20 cases is shown as Fig. 13. On this figure, a single shading indicates a faulty cell; arrowheads are attached to the two active inputs for that cell. A cell with cross-shading is assumed to be a good cell, and it replaces the faulty cell. If the faulty cell has no symbolism other than the arrowheads and the shading, it is assumed to have been disconnected by having all of cutpoints f, h, i, . . . , r open; if it has a J symbol, it is used as a jumper cell with no connections made at the arrowheads. Cells with a dotted single shading are neighboring good logical cells. Cases where the faulty cell in Fig. 13 occurs on or near the top row correspond to faults in $\alpha$ supercells, while cases where the faulty cell in Fig. 13 occurs on or near the bottom row correspond to faults in $\beta$ supercells.

From this development it should be clear that if a logical cell in a supercell array is bad, it can be logically replaced provided that another cell is good.

\[
\begin{align*}
R_{f} &= p_{0}T_{f} \\
R_{p_{i}} &= (\beta x_{i})T_{2} \\
R_{i} &= (y m_{i})T_{2} \\
R_{m_{i}} &= (a \beta p_{i})T_{2} \\
R_{q} &= f m_{0} T_{h_{0}} T_{2} \\
R_{h_{0}} &= (s y + T h_{0} y) T_{2} \\
R_{i_{1}} &= (l y + T h_{0} y) T_{2} \\
S_{i} &= p_{0}T_{i_{1}} \\
S_{p_{i}} &= t_{1} \\
S_{i} &= t_{1} \\
S_{m_{i}} &= t_{1} \\
S_{q} &= (f m_{0} + T h_{0} + a \beta)T_{2} \\
S_{h_{0}} &= t_{2} \\
S_{i_{1}} &= t_{2} \\
S &= f m_{0} \oplus T h_{0} \oplus q \\
x_{0} &= a \beta T_{3} h_{0}
\end{align*}
\]

Figure 14. Block diagram for a twelve-bit serial multiplier.
Thus a fault-avoidance algorithm for cobweb arrays has been demonstrated. Many variations in the process are possible. For instance, if multiple faults prevent complete avoidance of faulty cells using the $2 \times 2$ supercells, one can replace some or perhaps all cells in the supercell array again with supercells until enough redundancy has been obtained that all faults can be avoided. Similarly, it may be possible to compress a supercell array if, for instance, no corrections are necessary in a particular row or column. Similar fault-avoidance algorithms can be deduced for the simplified cobweb arrays mentioned before.

LOGICAL DESIGN OF A MULTIPLIER

As a final illustration, a logical design is given for a 12-bit serial multiplier in terms of a single cobweb cellular array. For comparison purposes, the same system has been chosen as was previously reported. The block diagram for this four-register, five-command multiplier is given as Fig. 14, while a previously-reported design in terms of five interconnected cutpoint arrays is shown in Fig. 15.

In Fig. 16, this same system is realized in terms of a single $27 \times 16$-cell cobweb array.

The statistics on these two realizations for the multiplier are as follows:

**Cutpoint Realization**

There are 352 cells in five cellular arrays, and 100 connections at the edges of the five arrays. 28% of the cells are "1" cells used only for transmitting information, 71% of the cells are used logically, and 1% of the cells are not used.

**Cobweb Realization**

There are 432 cells in one cellular array, and 10 connections at the edges of the one array. 26% of the cells are jumper cells, 55% of the cells are used logically, and 19% of the cells are not used.
It is seen from the above data that while 26 percent more cells are required for the multiplier in the cobweb realization than in the cutpoint realization, only one cobweb array is used versus five cutpoint arrays; furthermore, the backplane wiring in the cobweb realization is reduced by an order of magnitude.

CONCLUSIONS

The essential difference between the previously reported cutpoint cellular array and the cobweb array is the more complicated and flexible interconnection structure of the latter array. This flexibility allows the logical designer much more geometric freedom in the embedding of cascade logical realizations. For certain types of digital operations, and in particular for parallel operations, the use of cobweb arrays results in a significant reduction in the required number of cells. Also it is possible to eliminate jumper connections from one edge cell on an array to another edge cell on the same array when cobweb arrays are employed.

ACKNOWLEDGMENTS

The logical design of the parallel adder in Fig. 9 was provided by Mr. David W. Masters, the supercell concepts were the result of conversations with Mr. Milton W. Green, and some of the ideas on parallel operations in cellular arrays were derived from unpublished work of Mr. Jack Goldberg. Particularly credit is due to Dr. Robert A. Short for his critical evaluation of the manuscript and for his many helpful suggestions and comments.

Figure 16. Realization of the multiplier in terms of one cobweb array.
REFERENCES


