Universal logic circuits and their modular realizations

by S. S. YAU and C. K. TANG
Northwestern University
Evanston, Illinois

INTRODUCTION

In order to achieve the great economic advantage of utilizing integrated circuits in computer circuitry, it is desirable to design a circuit which can realize *any* logic function of a fixed number of variables by simply varying its input terminal connections. Such a circuit is called a *universal logic circuit (ULC)*. When the number of variables becomes large, a ULC may be too complex to be built in a single package economically. Hence, it is preferred to use ULC's of a small number of variables as the modules to build a ULC of a large number of variables. Such modules are called *universal logic modules (ULM's)*. In this paper, we shall first present a three-variable ULC, which has a fan-in for each logic gate not exceeding four, and consists of only 7 I/O pins. Then, we shall extend the ULC's to four or more variables. There are 12 I/O pins in a ULC of four variables, and several models with different fan-in limitations will be given. The logic gates in the ULC's may be all NAND or all NOR gates. Then, a simple technique for designing a ULC of any large number of variables using the ULC's of a small number of variables, say three variables, as the ULM's will be established. It will be seen that the ULC obtained by this technique will require a small number of ULM's. Moreover, the fault-detection tests for ULM's and a diagnostic procedure for locating all the faulty ULM's in the modular realization of a ULC realizing a given logic function will be presented. Finally, a method for improving the reliability of a ULC using an error-correcting code will be demonstrated.

Universal logic circuits of three variables

The problem of designing a ULC was first treated by Forslund and Waxman,[1] and later by Ellison, et al.[2] and Elspas, et al.[3] They employed the concept of equivalence classes to reduce the number of all possible logic functions of a given number of variables to the number of the equivalence classes. An equivalence class is a set of logic functions that may be obtained from a particular network by only manipulating the application of variables to the input terminals of the network. One of the most common constraints on these manipulations is that only true variables are available with the permutation of the variables at the input terminals permitted. With this restriction, Hellerman[4] partitioned the $2^{2^3} = 256$ three-variable logic functions into 80 equivalence classes. In order to reduce the number of equivalence classes, Forslund and Waxman[1] assumed that both true and complement variables are available at the input, and true and complement logic functions are both available at the output (two output terminals). In addition, biasing (to a logical 1 or 0) and duplication of input variables to the input terminals are also permitted. The equivalence classes defined this way reduces its number from 80 to 10 for three-variable logic functions. In this paper, the same constraints are to be placed on the manipulations of input terminals, except that only one output terminal will be required and that the biasing "0" and "1" are not necessary. We shall employ a different approach to obtain the ULC.

It is noted that a logic function f(x, y, z) of three variables x, y, z can always be expanded with respect to any two of the three variables x, y, z as follows:

$$f(x, y, z) = \bar{x}\,\bar{y}\,f(0, 0, z) + \bar{x}\,y\,f(0, 1, z) + x\,\bar{y}\,f(1, 0, z) + x\,y\,f(1, 1, z), \quad (1)$$

where the functions f(0, 0, z), f(0, 1, z), f(1, 0, z) and f(1, 1, z) are functions of z only, and each of these functions assumes one of the four values: z, z, 0 or 1. Hence, a circuit shown in Fig. 1 can realize any arbitrary three-variable logic function f(x, y, z) if the side terminals C_1 and C_2 are connected to x and y respectively and the four front terminals A_0, A_1, A_2, and A_3 are connected to the appropriate values z, z, 0, and 1. Based on (1) we obtain a ULC of three variables consisting of AND, OR and INVERTER gates shown in Fig. 1. It is noted that the biasing "0" and "1" are not necessary. In Fig. 1, for example, if f(0, 0, z) = 0, connecting terminal A_0 to biasing "0" is the same as connecting A_0 to input variable y;

and if f(0, 0, z) = 1, connecting terminal A_0 to biasing "1" is the same as connecting A_0 to input variable $\bar{y}$. Similar arguments apply to terminals A_1, A_2, and A_3.

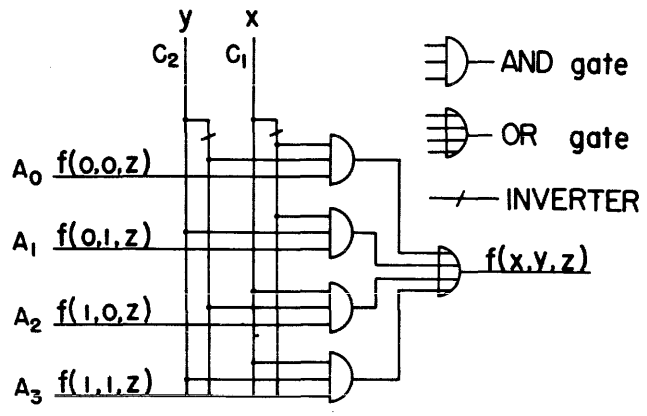

Figure 1 – A ULC of three variables consisting of AND, OR and NOT gates

It is well-known[5] that a two-level AND and OR circuit can be replaced by a NAND-gate circuit of the same configuration. Thus, the circuit shown in Fig. 2 is also a ULC of three variables employing NAND gates only.

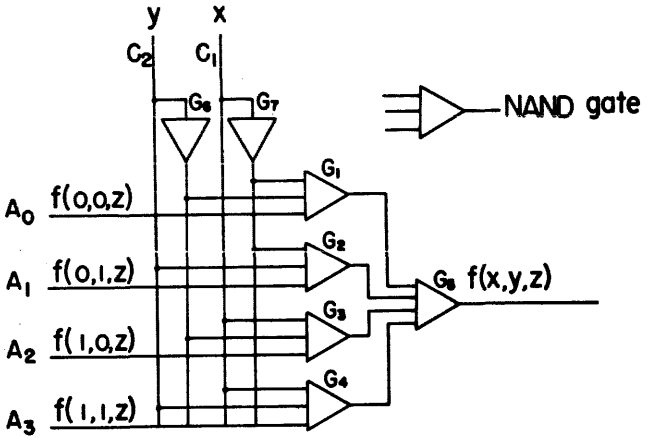

Figure 2 – A ULC of three variables consisting of NAND gates only

A ULC of three variables consisting of NOR gates can be obtained by using the dual relationship between NAND's and NOR's, and is given in Fig. 3. It is noted that the configurations of the ULC with NAND gates and the ULC with NOR gates are identical, and the only difference between these two circuits is the permutation of the input values for the front terminals. Another realization is shown in Fig. 4 using NAND, OR and INVERTER gates. This circuit is more desirable than those shown in Figs. 2 and 3, since it requires only 16 diodes and 3 transistors, while the circuits shown in Figs. 2 and 3 need 16 diodes and 7 transistors. Furthermore, the circuit shown in Fig. 4 is more reliable because fewer transistors are used. A similar circuit can be obtained using AND, NOR and INVERTER gates. In each of the above circuits, a total of 7 I/O pins is required.

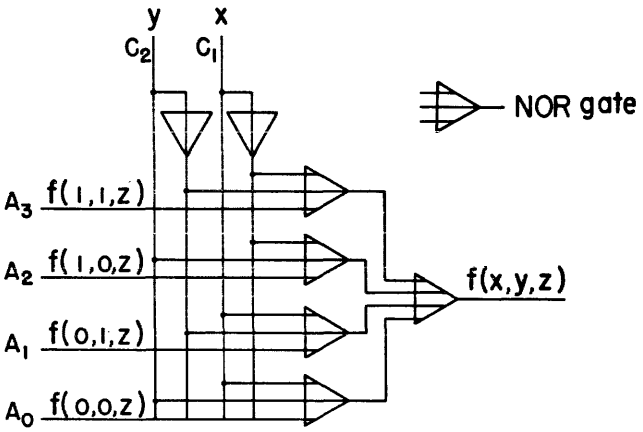

Figure 3 – A ULC of three variables consisting of NOR gates only

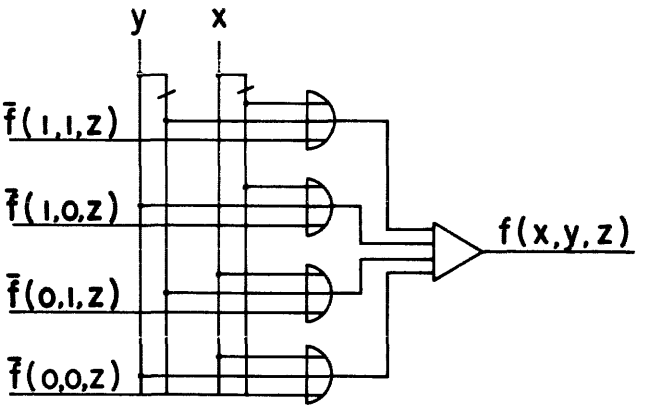

Figure 4 – A ULC of three variables consisting of OR, NAND and INVERTER gates

To evaluate the ULC given above, a comparison with the results given by Forslund and Waxman[1] is made as follows: Consider the circuit shown in Fig. 2 as a minimum-pin ULC. Since gates G_6 and G_7 are included in the ULC, the circuit has 7 pins, 7 gates, and 3 levels. The minimum-pin ULC of three variables given by Forslund and Waxman also has 7 pins, but it requires 10 gates and has 5 levels. The ULC given in Fig. 2 also has the advantage that only one complement input is required, whereas the minimum-pin ULC given by Forslund and Waxman requires two complement inputs. If the circuit shown in Fig. 2 is considered as a minimum-gate ULC, gates

G_6 and G_7 should be excluded from the ULC at the expense of adding two more pins. Consequently, it ends up with a minimum-gate ULC of 5 gates, 9 pins and 2 levels. The minimum-gate ULC of Forslund and Waxman has 6 gates, 9 pins and 4 levels, and can realize only the logic functions in nine out of the ten equivalence classes. It is seen that 5 is the absolute minimum number of gates required for any ULC of 3 variables, since the realization of the exclusive-or function of 3 variables alone requires a minimum of 5 NAND gates.[6]

Universal logic circuits of four and more variables

The problem of designing a ULC of four or more variables was also treated by Forslund and Waxman[1], using the same idea of equivalence classes as in the case of three-variable ULC. Due to the large amount of computations required, it is prohibitive to obtain such a ULC by that method. However, the approach used in the last section for obtaining the ULC of three variables can readily be extended to four or more variables. Since a logic function f(x, y, z, w) of four variables can be written in the form

$$\begin{aligned} f(x, y, z, w) = {} & \bar{x}\,\bar{y}\,\bar{z}\,f(0,0,0,w) + \bar{x}\,\bar{y}\,z\,f(0,0,1,w) \\ & + \bar{x}\,y\,\bar{z}\,f(0,1,0,w) + \bar{x}\,y\,z\,f(0,1,1,w) \\ & + x\,\bar{y}\,\bar{z}\,f(1,0,0,w) + x\,\bar{y}\,z\,f(1,0,1,w) \\ & + x\,y\,\bar{z}\,f(1,1,0,w) + x\,y\,z\,f(1,1,1,w), \end{aligned} \tag{2}$$

the ULC of four variables shown in Fig. 5 is obtained. It is noted that there is a NAND gate with a fan-in of 8 in this realization. Two other NAND realizations with smaller fan-in limitations and more gates are given in Figs. 6 and 7. Similar to the case of three-variable ULC's, the corresponding NOR realizations of Figs. 5-7 can easily be shown that they have the same configurations of the original NAND realizations with their input terminal connections permuted. The rule of permutation on the residue functions of one variable for the NOR realization is to replace 1 by 0 and 0 by 1 in the residue functions for a NAND realization. For instance, f(0, 1, 0, w) in Fig. 5 should be replaced by f(1, 0, 1, w) for the corresponding NOR realization.

The ULC's of five or more variables can be derived in a similar way. It can be shown that a ULC of n variables obtained by this method has p input pins, where

$$p = 2^{n-1} + n - 1. \tag{3}$$

With a fan-in limitation of four, this approach will yield a ULC of n variables, $n \geq 2$, which has q gates and ℓ levels, where

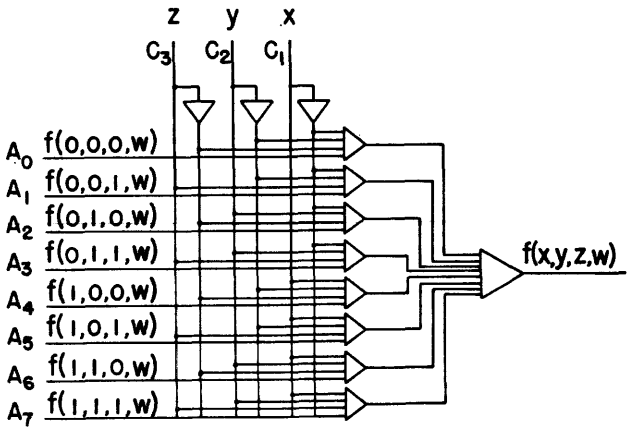

Figure 5 – A ULC of four variables with 3 levels and a fan-in 8

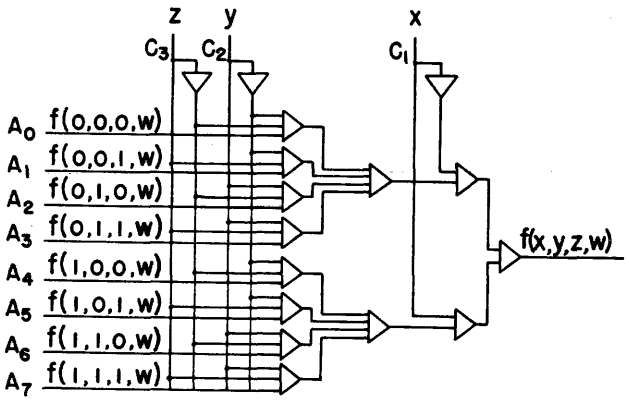

Figure 6 – A ULC of four variables with 5 levels and a fan-in 4

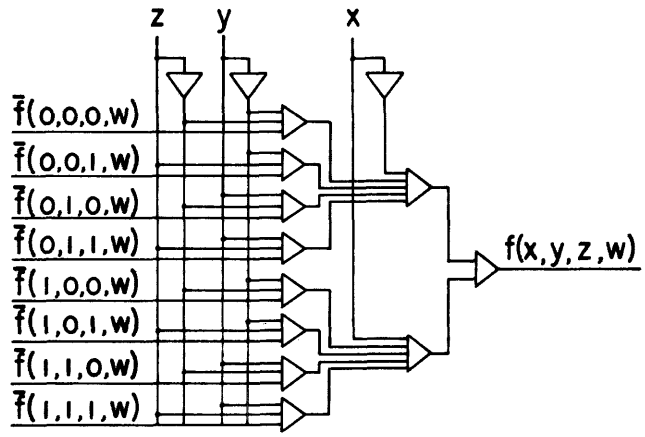

Figure 7 – A ULC of four variables with 4 levels and a fan-in 5

$$q = \begin{cases} \frac{5}{3}(2^{n-1}-1)+(n-1) & \text{when n is odd} \\ \frac{10}{3}(2^{n-2}-1)+(n+2) & \text{when n is even} \end{cases} \quad (4)$$

$$\ell = \begin{cases} n & \text{when n is odd} \\ n+1 & \text{when n is even.} \end{cases} \quad (5)$$

For example, a ULC of five variables with a fan-in limitation of four is shown in Fig. 8. The numbers given by (4) and (5) can certainly be reduced if a larger fan-in is permitted. It is noted that for any n only one complementary input variable is required and all others can be true input variables in a ULC obtained by this method.

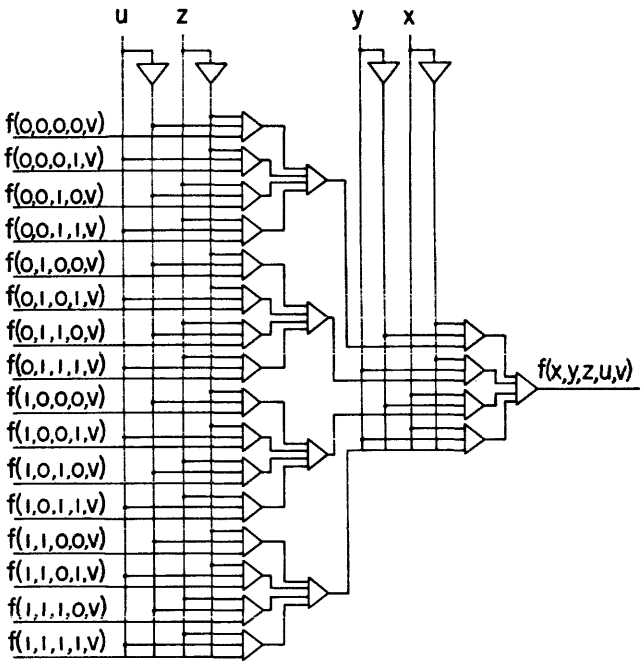

Figure 8 – A ULC of five variables

A comparison of the number of input pins required here and the upper bound of the number of input pins required for a ULC given by King[7] is shown in Table I. The upper bound given by King is for the ULC defined in a slightly different way, namely only true inputs are used, while in this paper one complementary input is allowed. It is seen that the values of p are considerably lower than King's upper bound. A lower bound on the number of required input pins is derived here with the restriction that only one complementary input is allowed. First we calculate the number of possible distinct connections of p input pins to the set of values $\{0, 1, x_1, x_2, \ldots, x_n, x_n\}$ such that every x_i, $1 \leq i \leq n$, is connected to at least one pin. It can be shown that this number is given by

The number p of input pins of a ULC and King's upper bound on p.

n	2	3	4	5	6
p	3	6	11	20	37
King's upper bound	6	11	20	37	70

TABLE I – The number p of input pins of a ULC and King's upper bound on p.

$$(n+3)^p - \binom{n}{1}(n+2)^p + \binom{n}{2}(n+1)^p - \binom{n}{3}n^p + \ldots + (-1)^n \binom{n}{n}3^p.$$

The number of possible distinct connections must not be smaller than the number of logic functions of n variables. Hence, the following inequality is obtained.

$$(n+3)^p - \binom{n}{1}(n+2)^p + \binom{n}{2}(n+1)^p - \binom{n}{3}n^p + \ldots + (-1)^n\binom{n}{n}3^p \geq 2^{2^n}.$$

The minimum p satisfying the inequality (6) is a lower bound on p, which is listed in Table II. It is noted that Elspas, et al,[3] have derived a ULC of 4 variables with a total of 9 I/O pins, and by decomposition a ULC of 5 variables with a total of 19 I/O pins was obtained. For $n \geq 6$, their result requires exactly the same number of input pins given by (3). They have also derived a lower bound for p, which is smaller than that listed in Table II, because all complementary inputs are allowed in their derivation.

TABLE II – The lower bound on p calculated according to (6).

n	2	3	4	5	6
Lower bound	3	5	7	12	21

Realization of a universal logic circuit using universal logic modules

We have shown in the last section that a ULC of any large number of variables can be found. However, it follows from (3)-(5) that the complexity of the ULC increases rapidly as the number of variables increases. From either economical point of view or maintenance point of view, it becomes prohibitive to build ULC's of various large numbers of variables in individual integrated circuit packages. Hence, we would like to present a technique for realizing a ULC of a large number of variables using identical ULC's of a small number of variables as modules, which are called universal logic modules (ULM's). Obviously, there are two great advantages of this technique. First, we only need a large quantity of identical

ULM's to build ULC's of various numbers of variables. Secondly, when there are faults in a ULC, we only need replace the faulty ULM's instead of the whole ULC.

To derive the modular realization of a ULC of n variables using ULC's of 3 variables as the ULM's (denoted by ULM-3's), let us first consider the case when n is odd. Since any logic function $f(x_1, x_2, \ldots, x_n)$ of n variables, $n \geq 3$, can be expanded to the form

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) = {} & \bar{x}_1\bar{x}_2 f(0, 0, x_3, \ldots, x_n) \\ & + x_1 x_2 f(0, 1, x_3, \ldots, x_n) \\ & + x_1\bar{x}_2 f(1, 0, x_3, \ldots, x_n) \\ & + x_2 x_2 f(1, 1, x_3, \ldots, x_n), \end{aligned} \tag{7}$$

it can be realized by a ULM-3, provided that the side terminals C_1 and C_2 and the front terminals A_0, A_1, A_2 and A_3 shown in Fig. 1, 2, or 3 are connected to the input variables x_1 and x_2 and the residue functions $f(0,0,x_3,\ldots,x_n)$, $f(0,1,x_3,\ldots,x_n)$, $f(1,0,x_3,\ldots,x_n)$ and $f(1,1,x_3,\ldots,x_n)$ respectively. This ULM-3 forms the first level of the modular realization of the ULC. Since we can repeat this process to each of the residue functions, the second level of the modular realization consists of four ULM-3's whose side terminals are connected to the input variables x_3 and x_4 and front terminals connected to appropriate residue functions of $n-4$ variables. Continue this process until the residue functions become functions of the variable x_n. Because n is odd, and because each expansion reduces the number of variables of the residue functions by exactly 2, it requires a total of $(n-1)/2$ expansions. This implies that $f(x_1, \ldots, x_n)$ can be realized by using ULM-3's in a tree structure consisting of $(n-1)/2$ levels as shown in Fig. 9. It is seen that there are 4^{j-1} ULM-3's in the j-th level of the tree structure. Each of the front terminals of the ULM-3's in the last level is connected to one of the four values 0, 1, x_n and $\bar{x}_n$ defined by the corresponding residue function of variable x_n, which can be found as follows: Trace the path from the output terminal F to the front terminal in the last level in question in the tree structure, and use two bits to write the binary representation of the subscript h for the front terminal A_h of the ULM-3 in each level. Then, the concentration of the $\frac{1}{2}(n-1)$ 2-tuples in the order of the path forms the argument of the residue function for the front terminal. For instance, if the path from the output terminal to a front terminal in the last level in a modular ULC of 5 levels passes through the front terminals A_1, A_2, A_0, A_3, A_1 of the ULM-3's in the 1st, 2nd,...,5th levels respectively, the residue function for this terminal is $f(0,1,1,0,0,0,1,1,0,1,x_n)$. For convenience, we shall call the front terminal of a ULM-3 in the last level P_i if it is connected to the residue function with the binary argument whose decimal representation is i. It is obvious that there are 2^{n-1} front terminals of the ULM-3's in the last level for a modular ULC of n variables. Fig. 10 shows the modular realization of a ULC of 7 variables using ULM-3's.

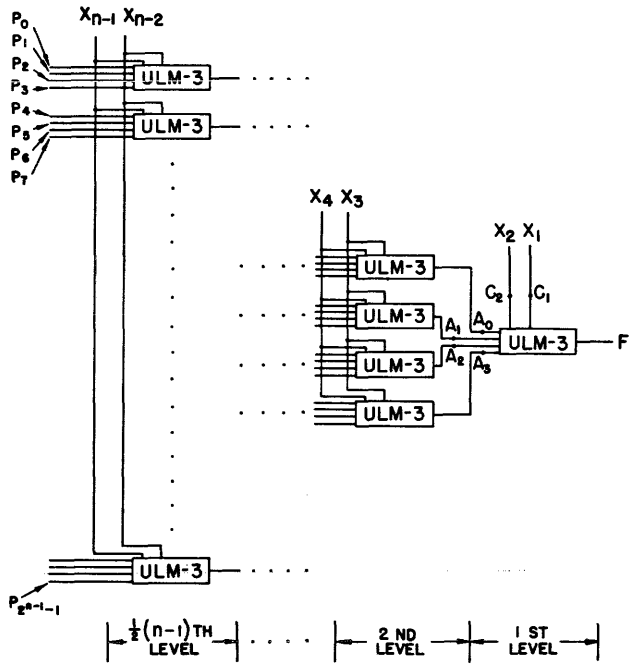

Figure 9—The modular realization of a ULC of n variables when n is odd

When n is even and when only ULM-3's can be used in the modular realization, only slight modification in the first level is required. Instead of expanding the logic function according to (7) for the first level, we only expand the logic function as follows:

$$f(x_1, x_2, \ldots, x_n) = \bar{x}_1 f(0, x_2, \ldots, x_n) + x_1 f(1, x_2, \ldots, x_n). \tag{8}$$

It is easily seen that (8) can be realized by a ULM-3, provided that the side terminals C_1 and C_2 are both connected to the input variable x_1, the front terminals A_0 and A_3 connected to the residue functions $f(0, x_2, \ldots, x_n)$ and $f(1, x_2, \ldots, x_n)$ respectively, and the connections for A_1 and A_2 are don't-care. Then, each of the residue functions in (8) is a function of an odd number of variables and hence can be realized by the previous method. The residue functions for the front terminals of the ULM-3's in the last level can be found in the same way as before except that only the first bit in the binary argument of the residue function corresponds to the subscript of the front terminal of the ULM-3 in the first level. The first bit is 0 or 1 depending upon whether the front terminal

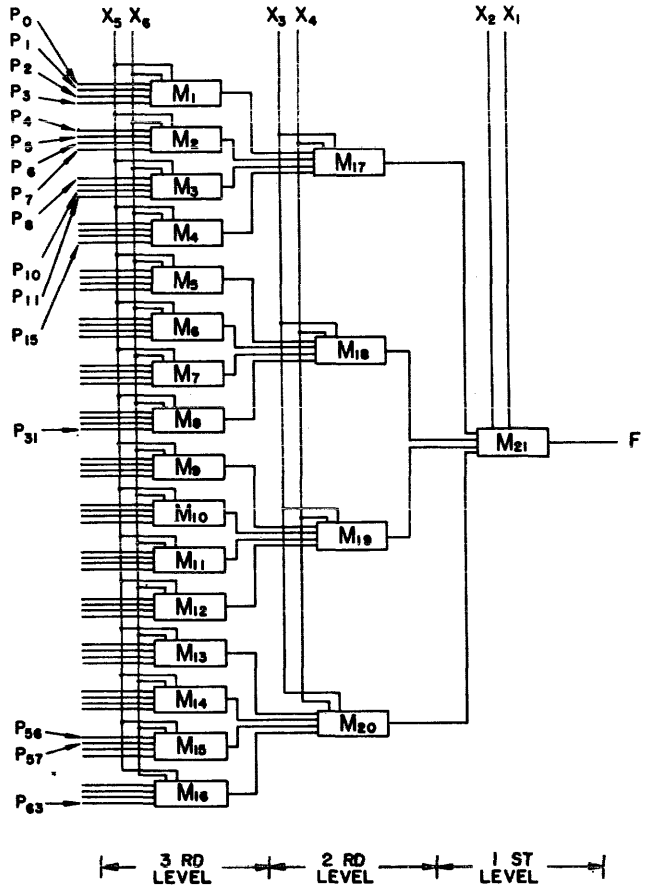

Figure 10—The modular realization of a ULC of 7 variables

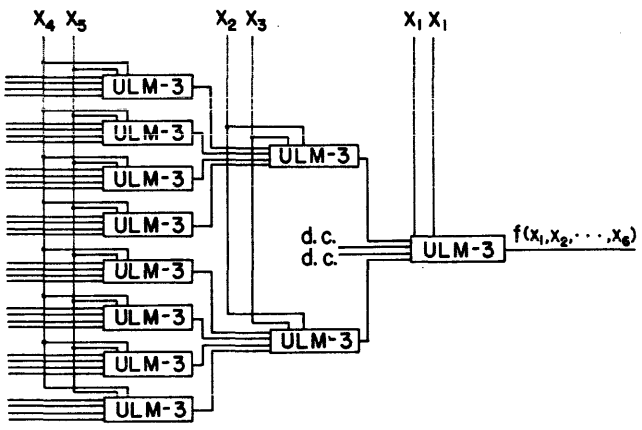

Figure 11—A modular realization of a ULC of six variables using ULM-3's only

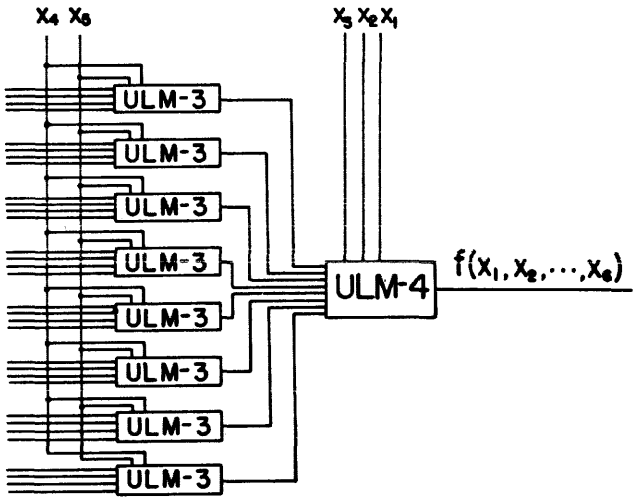

Figure 12—A modular realization of a ULC of six variables using ULM-3's and a ULM-4

of the ULM-3 in the first level in the path is A_0 or A_3 respectively. Fig. 11 shows such a modular realization of a ULC of 6 variables. It is noted that in this case we have not used the full capacity of the ULM-3 in the first level. In fact, if we are not restricted to use ULM-3's only in the modular realization, the three ULM-3's in the first and second levels can be substituted by a ULM-4 as shown in Fig. 12. The terminal connections and the residue functions for the front terminals of the ULM-4 in the last level can be found by considering the ULM-4 shown in Fig. 5 or 6 and the expansion of $f(x_1, x_2, \ldots, x_n)$ with respect to the variables x_1, x_2, and x_3.

The above modular realization technique can easily be extended to using ULM's of any variables. If only ULM's of a fixed number of variables can be used, it is often necessary to have some don't-care terminal connections to some of the ULM's and hence some of the ULM's are not utilized to their full capacity. It can be shown that if only ULM-4's are used in the modular realization, there will be no don't-care terminal connections to any ULM-4 for a ULC of n variables, where $n = 3\alpha + 4$ and α is a nonnegative integer. However, if both ULM-3 and ULM-4 are used in the modular realization, the don't-care terminal connections can always be avoided. Furthermore, it is noted that the tree structure of the modular realization of a ULC of n variables using ULM-k's always has 2^{n-1} front terminals in the last level for any k.

Fault-detection test for ULM's and a fault-diagnostic procedure for modular ULC's

The problem of developing a practical fault-diagnostic procedure for a logic circuit of a large number of variables is still far from being solved.[8-12] When integrated circuit packages are used for logical design,

so far there is no practical method of deriving a set of minimal fault diagnostic tests which can locate the faulty packages. At present, it is possible to find a set of tests to detect all faults and to locate *most* of the faults to within a reasonable number of packages.[13] In this section, we shall present the fault-detection tests for ULM's and a diagnostic procedure locating *all* the faulty ULM's in the modular realization of a ULC realizing a given logic function.

Fault-detection tests for ULM's

For a ULM built in an integrated circuit package, a defective unit usually means that a set of gates and possibly several connecting wires are burnt or broken. If we make no assumptions about the types of the faults in ULM's – whether they are due to single- or multiple-component failures, open- or short-circuit wires or gates, etc.—it is obvious that the fault-detection tests for ULM's must exhaust all possible combinations of the input terminals. Hence, for a ULM with p input terminals, where p is given by (3), it requires 2^p tests to detect all possible faults in a ULM. It is seen that a ULM-3 requires 64 tests* and a ULM-4 requires 2048 tests.

A diagnostic procedure to locate all the faulty ULM's in a ULC

Consider a ULC of n variables made of ULM-k's. Because a set of tests corresponding to all possible combinations of the n input variables will definitely detect all the faults in a logic circuit realizing a specific function, we need at most 2^n tests for detecting faults in the ULC realizing a given logic function.† If there are no restrictions on the type of possible faults in the ULM-k's, the 2^n tests also form the minimum test set.

Let a test applied to a ULC of n variables be represented by the n-tuple $(b_1, b_2, \ldots, b_n)$ of binary components, where b_ℓ is the value of the input variable x_ℓ, $\ell = 1, 2, \ldots, n$, employed in the test. Let T_i and T_i' be the tests $(b_1, b_2, \ldots, b_{n-1}, 0)$ and $(b_1, b_2, \ldots, b_{n-1}, 1)$ respectively, where $(b_1, b_2, \ldots, b_{n-1})$ has the decimal representation i. It follows from the tree structure of the modular realization of a ULC that if there are no faults in the ULC, the tests T_i and T_i will make the output terminal F *logically connected* to the front terminal P_i in the last level of the tree structure, where P_i is connected to the residue function $f(b_1, b_2, \ldots, b_{n-1}, x_n)$. Hence, the output terminal F and the terminal P_i should have the same value under the tests T_i and T_i'. When F and P_i have different values under test T_i or T_i' or both, the terminal P_i is said to *have a faulty test.* Furthermore, when we say that *apply tests to terminal* P_i, it implies that apply tests T_i and T_i' to the ULC. Because of the tree structure of the ULC, it is obvious that there is one and only one path from ohe output terminal F to each terminal P_i, and the path contains one and only one ULM-k in each level of the ULC. If a ULM-k M_r (of any level) is the paths terminating at the terminals $P_i, P_{i+1}, \ldots, P_{i+d}$, we shall say that M_r *covers* the terminals $P_i, P_{i+1}, \ldots, P_{i+d}$.

Now, the diagnostic procedure to locate all the faulty ULM-k's in a ULC of n variables can be summarized as follows:

1) Apply tests to terminals P_i, $i = 0, 1, \ldots, 2^{n-1}-1$. If there exists no terminals with faulty tests, go to Step 4);otherwise, go to the next step.

2) Start from $\delta = 1$. Let L_δ be the set of all the ULM-k's in the δth level in which each ULM-k covers at least one faulty terminal. Apply the fault-detection tests to each of the ULM-k's in L_δ. If all the ULM-k's in L_δ are good, go to the next step. Otherwise, replace each faulty ULM-k in L_δ and apply test to *all* terminals P_i's covered by each replaced ULM-k. Record the terminals P_i's with the new faulty tests and those terminals with previous faulty tests not covered by the replaced ULM-k's, and go to the next step.

3) Increase δ by 1, and go to Step 2) when δ is smaller than the number of levels of the tree structure. Otherwise, the ULM-k's (in the last level) covering at least one terminal with a faulty test are faulty and should be replaced, and go to Step 4).

4) All the ULM-k's in the ULC are good for realizing the logic function under consideration.

To demonstrate this diagnostic procedure, let us consider the ULC shown in Fig. 10. We first apply tests to all P_i's. Assume that terminals P_0, P_4, P_5, P_6, P_7, P_{31}, P_{56}, and P_{57}, be the terminals with faulty tests after replacing M_{17}. Hence, we know that P_1, P_8, and assume that we find M_{21} is good. Then, we have to apply fault-detection tests to M_{17}, M_{18} and M_{20} since each of these ULM-3's covers at least one terminal with a faulty test. Suppose we find that M_{17} is faulty and that M_{18} and M_{20} are good. Then, replace

*If the faults are restricted to "stuck-at-1" and "stuck-at-0" types, and if we assume that only a single fault can occur at a time in a ULM[9,11], it can be shown that the set of minimum tests for a ULM-3 consists of only 8 tests.

†It is noted that passing the 2^n tests only guarantees that the ULC will realize the logic function under consideration, but does not guarantee that the ULC will realize *any* logic function of n variables correctly. Similar to the fault-detection tests for a ULM, the set of tests required to ensure a ULC realizing any logic function of n variables correctly will have 2^q tests, where q is the number of input terminals of the ULC. However, if we assume that there is only a single faulty module in a ULC at a time, then the minimum number of tests required to ensure the ULC realizing any logic function of n variables correctly is 2^{n+3}.

M_{17} and apply tests to terminals $P_0,P_1,...,P_{15}$. Let P_1, P_8, P_{10} and P_{11} be the terminals with faulty tests after replacing M_{17}. Hence, we know that P_1, P_8, $P_{10},P_{11},P_{31},P_{56},P_{57}$ and P_{63} are all the terminals with faulty tests. Since we reach the last level, we know M_1,M_3,M_8,M_{15} and M_{16} are faulty and should be replaced. This terminates the procedure.

Improving the reliability of the modular realization of a ULC by an error-correcting code

The reliability of the modular realization of a ULC can be improved by adding redundant ULM's using an error-correcting code. In this section, we shall demonstrate how to apply Hamming single-error-correcting code to increase the reliability of the modular realization of a ULC of n variables. The circuit is shown in Fig. 13 and the following notations are employed.

$$\begin{aligned} f &= f(x_1,x_2,x_3,...,x_n) \\ f_0 &= f(0,0,x_3,...,x_n) \\ f_1 &= f(0,1,x_3,...,x_n) \\ f_2 &= f(1,0,x_3,...;x_n) \\ f_3 &= f(1,1,x_3,...,x_n) \end{aligned} \tag{9}$$

The four blocks B_0,B_1,B_2 and B_3 are the modular realizations of the ULC's of n-2 variables and have the outputs f_0,f_1,f_2 and f_3 respectively. The Hamming single-error-correcting code with 4 information symbols is used, and its parity-check matrix H and generator matrix G are given by

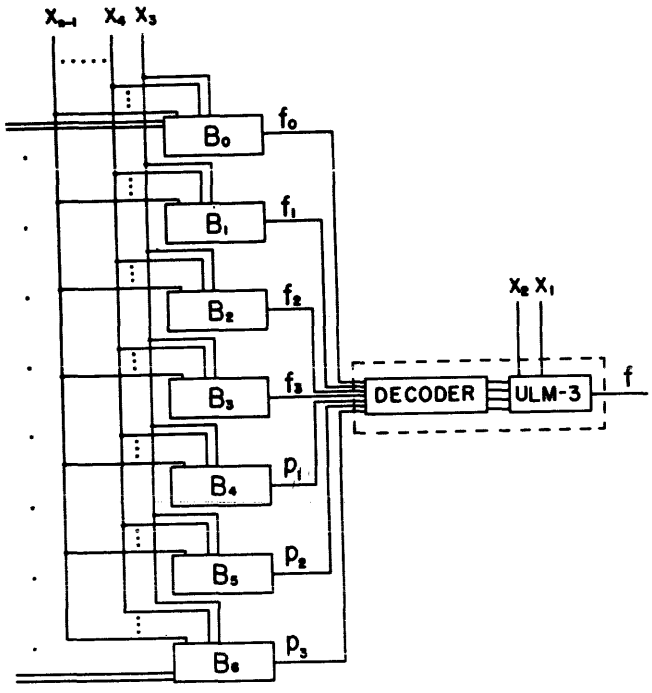

Figure 13—A modular realization of a ULC of n variables using Hamming single-error-correcting code

$$H = \begin{matrix} \\ \\ \\ p_1 \end{matrix} \begin{matrix} p_2 \end{matrix} \begin{matrix} f_0 \end{matrix} \begin{matrix} p_3 \end{matrix} \begin{matrix} f_1 \end{matrix} \begin{matrix} f_2 \end{matrix} \begin{matrix} f_3 \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{10}$$

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

The 4 information symbols to be encoded are f_0,f_1,f_2 and f_3, which are placed in the 3rd, 5th, 6th and 7th positions of the 7-bit code word respectively, while the remaining three positions are the parity check symbols p_1,p_2,p_3 as shown in (11). It follows from (10) and (11) that the parity-check symbols p_1,p_2 and p_3 can be expressed in terms of f_0,f_1,f_2 and f_3 as follows:

$$\begin{aligned} p_1 &= \bar{f}_0\bar{f}_1\bar{f}_3 + \bar{f}_0\bar{f}_1f_3 + f_0f_1f_3 + f_0\bar{f}_1\bar{f}_3 \\ p_2 &= \bar{f}_0\bar{f}_2f_3 + \bar{f}_0f_2\bar{f}_3 + f_0f_2f_3 + f_0\bar{f}_2\bar{f}_3 \\ p_3 &= \bar{f}_1\bar{f}_2f_3 + \bar{f}_1f_2\bar{f}_3 + f_1f_2f_3 + f_1\bar{f}_2\bar{f}_3 \end{aligned} \tag{12}$$

Since f_0,f_1,f_2 and f_3 are functions of the n-2 variables $x_3,...,x_n$, p_1,p_2 and p_3 are also functions of the same n-2 variables $x_3,...,x_n$. Thus, each of p_1,p_2,p_3 can be realized by using the modular realization of an ULC of n-2 variables. The three ULC's of n-2 variables for p_1,p_2 and p_3 are represented by the blocks B_4,B_5 and B_6 as shown in Fig. 13. The 7 signals f_0,f_1,f_2,f_3, p_1,p_2,p_3 are then fed to a decoder followed by a ULM-3 which produces the final output $f(x_1,\ldots,x_n)$. The decoder and the ULM-3 connected to the output terminal have to be of high reliability. It is found that the decoder will have 7 exclusive-OR gates, 3 INVERTER's and 4 AND gates. The decoder can be implemented together with the ULM-3 in a single reliable package. Let a block containing faulty ULM's be called a *faulty block*. It is seen that such a ULC of n variables will give correct output if there is only one faulty block. It is also noted that the ULC will still give correct output for the case that there are more than one faulty block, provided that only one erroneous block signal will show up at a time (under any input combination). If there exists a faulty block in the ULC, the easiest way to detect this faulty block is to add three output terminals to the decoder showing the syndrome of the code words. The faulty block can be located automatically

by simply reading the syndrome when the first fault occurs during the use of the ULC, and no separate test is required. The increase of cost for implementing this scheme is that for any $n > 3$, we have to add 75% redundant ULC's of n-2 variables and one highly reliable decoder-ULM-3 package. It is notes that the method illustrated above can easily be extended to the use of Hamming code with more than four information bits. Furthermore, the error-correcting code that can be used for increasing the reliability of the ULC is not restricted to the Hamming code, and the number of errors that can be corrected is not restricted to a single one.

DISCUSSION

In this paper, we have presented simple design techniques for ULC's, which are especially suitable to the use of integrated circuit packages for implementation. Various effects, such as the number of pins, the number of logic gates and the number of logic levels in a package, on the design of ULC's have been considered. For the ULC's obtained by the modular realization method, a diagnostic procedure for locating all the faulty ULM's in a faulty ULC has been established. Furthermore, a method for improving the reliability of a ULC using error-correcting codes has been demonstrated.

It is noted that an important practical advantage of using a ULC to realize a given logic function is that we need not find the minimal sum or minimal product of the logic function, which, however, is required for conventional realization methods. The only simplification process necessary to be applied to the logic function is to detect whether it can be written in a form which involves fewer variables. This result is used to determine a ULC of the smallest number of variables for realizing the given logic function.

It should be pointed out that the ULC's considered in this paper are restricted to realizing any single logic function. A natural extension of this study is to consider the design of a multiple-output ULC for realizing any set of m logic function. One way to obtain such a multiple-output ULC is simply to connect the m ULC's, each of which realizes one of the m logic functions, in the form of sharing the common input-variable terminals. It is quite unlikely that a multiple-output ULC with fewer I/O terminals can be obtained, because in general there are no fixed relations among the m logic functions to be realized.

ACKNOWLEDGMENT

The work reported here was supported in part by the U. S. Office of Scientific Research under Grant No. AF-AFOSR-1292-67.

REFERENCES

1 D C FORSLUND R WAXMAN
The universal logic block (ULB) and its application to logic design
Conference Record of 1966 Seventh Annual Symposium on Switching and Automata Theory IEEE Publication 16C40 pp 236-250

2 J T ELLISON B KOLMAN A P SCHIAVO
Universal function modules
UNIVAC Tech Rept Contract No AF19(628)-6012 (DDC AD-655395) April 1967

3 B ELSPAS et al
Properties of cellular arrays for logic and storage
Stanford Research Institute Scientific Report 3 Contract No AF-19-628-5828 (DDC AD-658832) pp 59-83 June 1967

4 L HELLERMAN
A catalogue of three-variables OR-INVERT and AND-INVERT logical circuit
IEEE Transaction on Electronic Computers vol 12 pp 198-223 1963

5 R B HURLEY
Transistor logic circuits
New York Wiley 1961

6 R A SMITH
Minimal three-variable NOR and NAND logic circuits
IEEE Transactions on Electronic Computers Vol EC-14 No 1 pp 79-81 February 1965

7 W FRANK KING III
The synthesis of multipurpose logic devices
Conference Record of 1966 Seventh Annual Symposium on Switching and Automata Theory IEEE Publication 1640 pp 227-235

8 J D BRULE R A JOHNSON E KLETSKY
Diagnosis of equipment failures
IRE Transactions on Reliability and Quality Control vol ol RQC-9 pp 23-34 1960

9 J M GALEY R E NORBY J P ROTH
Techniques for the diagnosis of switching circuit failures
IEEE Transactions on Comm and Elect Vol 83 No 74 pp 509-514 1964

10 H Y CHANG
An algorithm for selecting an optimum set of diagnostic tests
IEEE Transactions on Electronic Computers Vol EC-14 No 5 pp 705-711 1965

11 D B ARMSTRONG
On finding a nearly minimal set of fault detection tests for combinational logic nets
IEEE Transactions on Electronic Computers vol EC-14 no 1 pp 66-73 1966

12 W H KAUTZ
Fault diagnosis in combinational digital circuit
First Annual IEEE Computer Conference Digest IEEE Publication 16C51 pp 2-5 1967

13 *Logic partitioning in LSI*
Panel Discussion L M Spandorfer (moderator) IEEE Computer Group News vol no 6 p 16 May 1967