The use of Roth's decomposition algorithm in multi-level design of circuits

By D. F. Barnard and D. F. Holman*

The paper describes a procedure for synthesising multi-level combinatorial switching circuits. This procedure is based on an algorithm of Roth and Karp to find the possible decompositions of a partial switching function. The procedure builds up a circuit using a specified set of gates of vertex type. By repeated application of the algorithm, a complete realisation of the function is obtained whose cost is defined to be the sum of the costs of the basic elements. This cost can then be used as a cost bound in a systematic search for a realisation of minimum cost.

(First received January 1968 and in revised form May 1968)

The synthesis of switching circuits has received a good deal of attention from many authors, particularly in the realm of 2-level minimisation using prime implicant tables or charts. However, much work has also been done on other aspects of the problem, particularly from the point of view of decomposition theory (Ashenhurst, Roth), and it is this line of attack which has led to the computer applications described herein. A decomposition algorithm was published by Roth and Karp (1962) and some computer programming was done in America in the early sixties. The present authors have made use of this algorithm as the basis of a KDF9 computer program to produce a realisation of a given logical function from a given set of logical units. The intention is to obtain a circuit which is economical. If the algorithm is carried to completion, a circuit of minimum cost will be obtained; but the computation time for this may be excessively long if the number of variables is large.

Decompositions and the synthesis problem

This section deals with the basic ideas behind the algorithm and is purely expository.

A Boolean function F of n variables is frequently specified by a table of combinations. The n variables are associated with the inputs to a black box. The table lists the combinations of the values of the variables, either 0 or 1, for which $F = 1$, i.e. the black box has an output. If don't-care conditions are also specified it becomes necessary, for the purposes of explaining the algorithm, not only to have the above list but also a table of input combinations for which $F = 0$. If there are any input combinations not included in these two lists they are the don't-care ones. The first table is called the *on-matrix*, the second the *off-matrix*.

An example of a Boolean function of (say) four variables might be:

ON-MATRIX	OFF-MATRIX
0111	1100
0101	1010
1011	1001
1101	0011

A more compact notation is achieved by using n-tuples of 0, 1 and x. These n-tuples we call *cubes* and if no x is present the n-tuple is called a *vertex*. A cube is a shorthand notation for a set of vertices obtained from the cube by changing the xs to 0 and 1 in all possible ways, e.g. 1x01 stands for 1001 and 1101.

A five-variable Boolean function might then be specified in the following way:

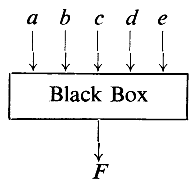
ON-MATRIX	OFF-MATRIX
abcde	*abcde*
1x11x	10000
00x11	100x1
011x1	1x01x
00000	

The on-matrix corresponds to the usual sum of product form of a Boolean function, i.e. in the above example

$$f = acd + \overline{ab}de + \bar{a}bce + \overline{abcde}$$

Now consider the circuit of **Fig. 1.** It may have been built up by the following process:

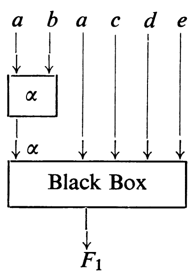
(i)



This represents the given specification

$$F = F(a, b, c, d, e).$$

(ii)



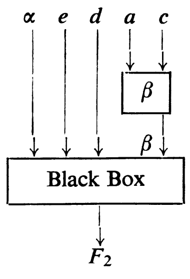
* *Mechanical Engineering Laboratory, The English Electric Company Limited, Whetstone, Nr. Leicester.*

By tests to be explained later, the black box of (i) could be replaced by the above circuit, i.e.

$$F(a, b, c, d, e) = F_1(\alpha(a, b), a, c, d, e).$$

The move from stage (i) to (ii) is called a *decomposition* of the given Boolean function F w.r.t. α and the given partition of variables, and F_1 is called the *image* of the decomposition. Assuming that such a decomposition exists one has next to find the on and off matrices of F_1 (see later)

(iii)

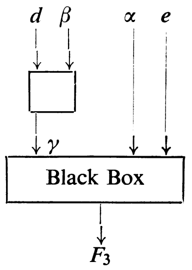


At this stage we have

$$F_1(\alpha, a, c, d, e) = F_2(\beta(a, c), \alpha, e, d)$$

This equation represents a decomposition of the function F_1, and F_2 is the image.

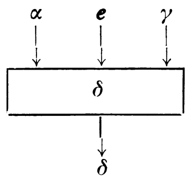
(iv)



Next we have

$$F_2(\beta,\alpha, e, d) = F_3(\gamma(d, \beta), \alpha, e)$$

(v)



This final stage gives

$$F_3(\gamma, \alpha, e) = \delta(\gamma, \alpha, e).$$

Piecing together this sequence of decompositions we have

$$F(a, b, c, d, e) = \delta(\gamma(d, \beta(a, c)), \alpha(a, b), e)$$

representing the circuit of Fig. 1. Each decomposition corresponds to 'factoring off' an element of the circuit.

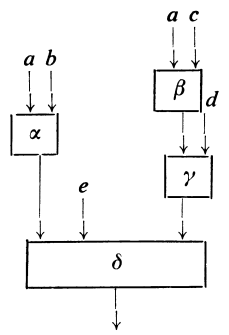


Fig. 1. Realisation of a function by single output elements

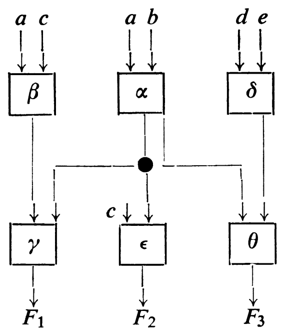


Fig. 2. Multi-output function, α is an element with 2 outputs

The sequence is of course not unique as we could have started by 'factoring off' the element β, etc.

Similarly it can be shown that **Fig. 2** corresponds to the decompositions:

$$F_1 = \gamma_1(\alpha_1(a, b), \beta_1(a, c))$$
$$F_2 = \epsilon_1(\alpha_1(a, b), \delta_1(d, e), c)$$
$$F_3 = \zeta_1(\alpha_2(a, b), \delta_2(d, e)).$$

Decomposition tests

Consider a function of the n variables $\{x_1, \ldots, x_n\} = X$. Partition X into 2 disjoint subsets (no elements in common) Y and Z such that every x_k is either in Y or in Z. Let $Y = \{y_1, \ldots, y_s\}$ and $Z = \{z_1, \ldots, z_{n-s}\}$; the variables of Y and Z are the variables of X, possibly in a different order. If $F(X) = G(\alpha(Y), Z)$, then $G(\alpha, Z)$ and $\alpha(Y)$ are said to represent a simple disjunctive decomposition of $F(X)$. It is said to be disjunctive because Y and Z are disjoint.

Now $F(X)$ is given by on and off matrices. We may rearrange the variables of X so that the variables of Y occur first and are followed by the variables of Z. The first part of a cube in this rearranged matrix is called the λ-part and the remainder the μ-part.

As an example let $F(a, b, c, d)$ be given by:

	a	b	c	d		a	b	c	d
u^1:	1	0	1	x	v^1:	1	1	x	1
u^2:	1	x	1	0	v^2:	1	x	0	x
u^3:	0	1	x	x	v^3:	x	0	0	x

Refer to the cubes in the on-matrix as u^1, u^2, u^3 and those in the off-matrix as v^1, v^2, v^3.

Now if $\lambda = \{a, b\}$, $\mu = \{c, d\}$ then no rearrangement of variables is necessary and e.g. $u^1_\lambda = 10$, $v^2_\mu = 0x$.

In order to decide whether there is a decomposition $F(X) = G(\alpha(Y), Z)$, we introduce the idea of incompatibility. Any two cubes u and v, one in the on-matrix and one in the off-matrix, must have different images as the result of the decomposition $F(X) = G(\alpha(Y),Z)$. Therefore if the two cubes have a common μ-part, we must have $\alpha(u_\lambda) \neq \alpha(v_\lambda)$, for otherwise some cubes of the image $G(\alpha, Z)$ would appear in both the on- and off-matrices. We then say that the λ-parts of u and v are incompatible if their μ-parts intersect. λ-parts which are not incompatible are said to be compatible.

In the example, u^1_μ and v^1_μ have the intersection 11. Since $F(1011) = 1$ and $F(1111) = 0$, it is necessary that $G(\alpha(10), 11) \neq G(\alpha(11), 11)$, and so $\alpha(10) \neq \alpha(11)$; in other words the λ-parts 10 and 11 are incompatible, written $10 \nsim 11$.

In order to determine whether a decomposition exists, we consider every pair of cubes u^i, v^j. If their μ-parts intersect we obtain an incompatibility. If all the incompatibilities are taken into account, we must be able to divide all the λ-parts into 2 mutually compatible sets in order to obtain a decomposition. Then one of these sets will have $\alpha(Y) = 1$, the other $\alpha(Y) = 0$.

Referring again to our example, we have the following situation,

u^1_μ intersects v^1_μ, giving $10 \nsim 11$

u^3_μ intersects v^1_μ, giving $01 \nsim 11$

u^3_μ intersects v^2_μ, giving $01 \nsim 1x$

u^3_μ intersects v^3_μ, giving $01 \nsim x0$

The λ-parts cannot be split into fewer than 3 mutually compatible sets, since $10 \nsim 11$, $01 \nsim 11$ and $01 \nsim 10$; therefore there is no decomposition for any α, since we must assign 3 different values to $\alpha(10)$, $\alpha(11)$ and $\alpha(01)$, and we have only the 2 values 0 and 1 available.

In the foregoing remarks we have restricted ourselves to single-output elements α; Roth's algorithm covers also the situation when multiple outputs are allowed, with decompositions of the form

$$F(X) = G(\alpha_1(Y), \alpha_2(Y) \ldots \alpha_r(Y), Z),$$

the condition for existence of a decomposition is that the number of mutually compatible sets must be $\leqslant 2^r$. Thus in the example above a decomposition can be found if we use elements with 2 outputs.

These considerations are stated formally in two theorems in the paper of Roth and Karp (1962).

Theorem 1

Given F and α, there exists G such that $F(Y, Z) = G(\alpha(Y), Z)$ if and only if, for all $y_1 \epsilon Y$ and $y_2 \epsilon Y$, $y_1 \nsim y_2$ implies $\alpha(y_1) \neq \alpha(y_2)$.

		ab 00	01	10	11
cd	00	0	1	0	0
	01	0	1	0	0
	10	x	1	1	1
	11	x	1	1	0

ON-MATRIX		OFF-MATRIX	
λ	μ	λ	μ
ab	*cd*	*ab*	*cd*
10	1x	11	x1
1x	10	1x	0x
01	xx	x0	0x

Fig. 3. Decomposition chart—disjunctive case

		ab 00	01	10	11
bcd	000	0	x	0	x
	001	0	x	0	x
	010	x	x	1	x
	011	x	x	1	x
	100	x	1	x	0
	101	x	1	x	0
	110	x	1	x	1
	111	x	1	x	0

ON-MATRIX		OFF-MATRIX	
λ	μ	λ	μ
ab	*bcd*	*ab*	*bcd*
10	01x	11	1x1
11	110	11	10x
01	1xx	x0	00x

Fig. 4. Decomposition chart—non-disjunctive case

Theorem 2 (stated here in simplified form)

If the cubes of Y can be partitioned into not more than two classes of mutually compatible elements then there exist α and G such that $F(Y, Z) = G(\alpha(Y), Z)$.

Up to now we have been talking about disjunctive decompositions. If the subsets Y, Z are not disjoint then the decomposition $F(Y, Z) = G(\alpha(Y), Z)$ is said to be non-disjunctive and the variables $Y \cap Z$ are referred to as the redundant variables. The non-disjunctive case is easily dealt with by including the redundant variables in both λ- and μ-parts; e.g. if $\lambda = (a, b)$, $\mu = (b, c, d)$ our example becomes

a b b c d	*a b b c d*
1 0 0 1 *x*	1 1 1 *x* 1
1 *x x* 1 0	1 *x x* 0 *x*
0 1 1 *x x*	*x* 0 0 0 *x*

Where an x occurs in the b position we must exclude cubes which require b to be both 0 and 1 simultaneously. The term 1 x x 1 0 on the left is therefore replaced by the 2 terms 1 1 1 1 0 and 1 0 0 1 0, and on the right 1 x x 0 x is replaced by 1 1 1 0 x and 1 0 0 0 x. We have now reduced the problem to the disjunctive case.

The notion of compatibility has a straightforward interpretation in terms of decomposition charts. The example discussed above can be represented in terms of the chart of **Fig. 3**.

The λ-parts appear at the head of the column, and the μ-parts at the beginning of the row. Then if two columns of the chart are the same, apart from entries where the function is undefined, then the vertices heading these columns are compatible, and the incompatibilities are seen as distinct columns in the chart. In fact, the three incompatibilities we found can be seen as the three different columns 01, 10, and 11. Theorem 2 translated means that the columns must be able to coalesce into not more than two distinct columns in order to obtain a decomposition. The subject has been treated from the point of view of charts by Curtis (1962), but this is hardly suitable for computer implementation. It appears that in fact we now have the algebraic equivalent of a chart technique. The redrawing of the chart with different variables appearing as rows and columns corresponds directly in our method to changing to a different λ-part and μ-part. The interpretation of the non-disjunctive choice of λ and μ as mentioned above is shown in **Fig. 4**. It can be seen that it is now possible to find a decomposition; in fact there is only one incompatibility relation, $01 \nsim 11$, so that we can choose any function α for which $\alpha(01) \neq \alpha(11)$.

Decomposition tables

A tabular method for determining decompositions of vertex type will now be described. A vertex function is one which is either ON for only one input combination, or is only OFF for one input combination. This includes the commonly used gates such as AND, OR, NAND, NOR, but excludes non-equivalence. The input combination which is so distinguished is called the distinguished vertex, $V\alpha$. A decomposition $F(X) = G(\alpha(Y), Z)$ is said to be of vertex type if α is a vertex function.

Suppose that we have a 4-input function $F(a, b, c, d)$ with inputs a, b, c, d and that the cubes $u = 1001$, $v = 1101$ appear in the ON, OFF arrays respectively. Notice that these two cubes differ in the input b but have the same input values for a, c, d. Now an incompatibility relation will be obtained from these two cubes if and only if we choose the μ-parts so that they have an intersection; i.e. if b is not in the μ-part but in the λ-part. If we attempted all possible disjunctive 2-input vertex decompositions in turn, then from consideration of these two cubes we should obtain incompatibility relations for decompositions with λ-parts (a, b), (b, c) and (b, d) but not for the λ-parts (a, c), (a, d) and (c, d). These incompatibilities are obtained by simply writing down the λ-parts concerned, for example, for $\lambda = (a, b)$ we get $11 \nsim 10$. All the incompatibilities can be expressed at once by writing down all the possible distinguished vertices in the form

$$\begin{array}{cc} & a\ b\ c\ d \\ (b) & 1\ 1\ 0\ 1 \\ & 1\ 0\ 0\ 1 \end{array}$$

where it is to be emphasised that b has been assumed to be in the λ-part.

Suppose now that the ON and OFF arrays also contain the cubes $01x1$, 0001 respectively. These also differ in the input b, and so another set of incompatibilities arises when we consider all λ-parts which include b, expressible in a form similar to the above as

$$\begin{array}{cc} & a\ b\ c\ d \\ (b) & 0\ 1\ X\ 1 \\ & 0\ 0\ 0\ 1 \end{array}$$

Consider now the total effect of all these incompatibilities. If $\lambda = (a, b)$, we have $11 \nsim 10$ and $01 \nsim 00$. Hence there can be no distinguished vertex. If $\lambda = (b, c)$ we have $10 \nsim 00$ and $1\,x \nsim 00$; hence there is a distinguished vertex, 00. If $\lambda = (b, d)$, we have $11 \nsim 01$ (twice) so either of these may be a distinguished vertex. Our shortened way of expressing all these results is

$$\begin{array}{cc} & a\ b\ c\ d \\ (b) & X\ 1\ X\ 1 \\ & X\ 0\ 0\ 1 \end{array}$$

Now this can be obtained by combining in pairs the cubes from the two expressions above, and putting X where either contains an x, and also where the two corresponding cubes are opposed
i.e.

$$\begin{array}{lllll} (b) & 1101 & 01X1 & & X1X1 \\ & 1001 & \text{and}\ 0001 & \text{gives} & X001 \end{array}$$

Define the distance between two cubes as the number of input positions at which one cube contains a 1 while the other contains a 0. The distance between the pairs above was 1.

Take two cubes at distance 2, e.g. 0101,0000. Unless we require α to have inputs b, d there are no incompatibilities. With $\alpha = \alpha(b, d)$ we have $V\alpha = 11$, or 00. Thus for pairs at distance 2 we make a partial entry

$$\begin{array}{lllll} (b) & X1X1 & .1.1 & & X1X1 \\ & X001 & \text{and}\ .0.0 & \text{gives} & X00X \end{array}$$

Pairs at distance 3 or more do not give rise to any incompatibilities. The restrictions for variable b now state that the only vertex-type decompositions available are

λ	μ	$V\alpha$	α
b, d	a, c	11	bd or $\overline{bd}$
b, c	a, d	00	$b+c$ or $\overline{b+c}$

Thus we have an algorithm with which to produce a table of all possible 2-input disjunctive decompositions. The table is composed of a number of subtables, one for each variable, i.e. the ith subtable indicates those decompositions allowed with the ith variable as an input. These are the ones that have not been eliminated by the above process.

When we choose a decomposition with inputs i, j from the ith subtable, we must check that this is also contained in the jth subtable.

In **Fig. 5** we have drawn up the complete decomposition table for the function specified in Fig. 3. The a-subtable indicates a decomposition, λ-part (a, b) with $V_\alpha = 01$ but this is not confirmed in the b-subtable and so is invalid. Note that it is only necessary to look on the right-hand side of the main diagonal for possible decompositions, the left-hand side being used to confirm the decomposition in the second subtable. Thus the table indicates no disjunctive decompositions.

The algorithm consists of the following steps

(1) Take a pair of cubes from the specification, one from the ON array, one from the OFF array.
(2) Find the distance, d between them.
(3) If $d = 0$ fail.
(4) If $d = 1$ a complete entry is made in the appropriate subtable.
(5) If $d = 2$ a partial entry is made in the subtable corresponding to the left-hand variable.
(6) If $d \geqslant 3$ the pair is ignored.
(7) Go to (1).

This is continued until all pairs have been investigated.

1-redundancy decompositions

We can also draw up a table of decompositions with one variable redundant. The difference between this case and the disjunctive one is that now, a pair of cubes only restricts the choice of α if they are at distance 1, because otherwise the μ-parts do not intersect.

E.g. take one of the previous pairs $01x1$, 0001; if the inputs are a and b or b and d then the restrictions are as before, but for inputs b, c 0111, 0001 are at distance 2 and so do not contribute anything, while the pair 0101, 0001 indicate $V_\alpha = 10$, or 00.

So the restrictions of $01x1$, 0001 are written 0–01 where the dash indicates that either a 1 or 0 may be used.

Another pair of terms $x100$, $00xx$ would be summarised as 0–00. The pairs of entries are thus combined as follows

$$\begin{array}{ccc} 01x1 & & x100 \\ & , & \\ 0001 & & 00xx \end{array} \rightarrow 0\text{–}01,\ 0\text{–}00 \rightarrow 0\text{–}0x$$

Thus whereas each subtable of the disjunctive table contains two entries, the subtables in the 1-redundancy case only contain one. The ith subtable of the 1-redundancy table indicates which decompositions are allowed with the ith variable as the non-redundant input. **Fig. 6** shows the decomposition table for the function of Fig. 3.

We can now modify our algorithm so that at the fourth step, if the distance is 1 then we make an entry at the appropriate subtables of both the disjunctive table and the 1-redundancy table.

Procedure for synthesis of a function

The procedure for obtaining a complete realisation is as follows. The starting point is the on- and off-matrices of the function, and the relative costs of the available gates; the basic steps needed to achieve a circuit of minimum cost are

(1) determine what possible decompositions exist
(2) select a decomposition and find its image
(3) find the cost of the circuit at this stage (this is defined as the sum of the costs of the elements which make up the circuit).

The repetition of these steps continues until the function is reduced to a single variable, when we have a complete circuit of known cost. It may then be thought desirable to try to find other realisations which may be cheaper. This can happen if we have at any time made a choice at step 2 of decomposition which is not the best. In order to improve the situation we can then retrace our steps and try a different choice. The search may be continued as long as it is thought worth while—to try every possible circuit may of course take some considerable time, and it is usually better to be content with a reasonably good circuit.

Returning to our example, let us try and complete the realisation. We already know from Fig. 4 that there are non-disjunctive decompositions with $\lambda = (a, b)$, $\mu = (b, c, d)$. Selecting $\alpha = \bar{a}.b$ as the first choice, we get the image

α	b	c	d
0	0	1	x
0	1	1	0
1	1	x	x

α	b	c	d
0	1	x	1
0	x	0	x

The only disjunctive decomposition available is $\beta = b.d$,

	a	b	c	d
a	1	X	X	X
	0	1	X	X
b	X	1	X	X
	X	0	X	X
c	1	X	1	X
	X	X	0	X
d	1	1	X	1
	1	X	1	0

Fig. 5. Decomposition table for the example of Fig. 3, disjunctive case

	a	b	c	d
a	—	1	X	X
b	X	—	X	X
c	1	X	—	X
d	1	1	1	—

Fig. 6. Decomposition table for the example of Fig. 3, 1-redundancy case. From the a-subtable we see that there are two decompositions with redundant variable b, $V_\alpha = 01$ or 11. The table indicates a total of ten possible decompositions.

giving the image

α	c	β
0	1	0
1	x	x

α	c	β
0	x	1
0	0	x

Again there is only one disjunctive decomposition, $\gamma = c\bar{\beta}$. The image is

α	γ
0	1
1	x

α	γ
0	0

This is a simple OR function, so we have the complete circuit, shown in **Fig. 7.** This compares with the normal form realisation in **Fig. 8** which can easily be obtained from the chart of Fig. 3.

Computer implementation on KDF9

The algorithm has not been programmed in its full generality. At present single-output decompositions only are included, and the decompositions are restricted to those of 'vertex type'. At present two-input gates only are allowed; these are represented by the decompositions with two variables in the λ-part. The given function may have up to 47 variables. The flowchart for the program is shown in **Fig. 9.**

Because of the length of time required to search out a large problem completely it is desirable that the first circuit generated should be reasonably economic. This is important even if a complete search is anticipated, as the cost of the last circuit found is used as a cost bound to limit the search, e.g. if at any stage there are n inputs to the image then the cost of a circuit using this image must be at least

$$(n - 1) \times \text{minimum gate cost} + \text{cost so far}$$

and if this is greater than the cost bound then we can reject this sequence. Ideally we would feed in the cost of the cheapest circuit as data and use this as our cost bound.

The rules that we use to select decompositions are as follows:

(1) Basically we select them in the order in which they appear in the table.
(2) If at any stage we have to select a 1-redundancy decomposition we use a '1-step look ahead' procedure to see if there is one which will allow us to to do a disjunctive decomposition at the next stage.
(3) If there is a cost bound then we do the cost check outlined above.
(4) It is necessary to prevent the phenomenon known as cycling. This is the term used when the output from a gate is either a constant or a function of only one of its inputs.
e.g. the decomposition $\alpha(a, b) = a.b$ followed by $\beta(\alpha, b) = \alpha + b$ means that the output from β is b.

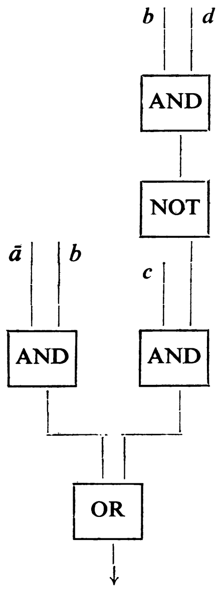


Fig. 7. The example in the text, solved by Roth's algorithm

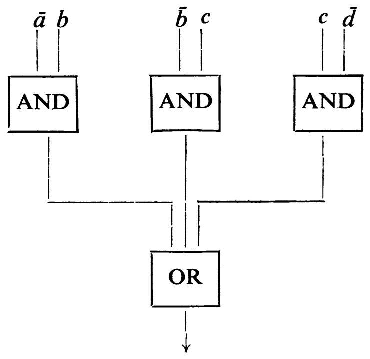


Fig. 8. Sum of products realisation of the same example; this is more expensive if complemented inputs are not available.

(5) It is desirable to prevent the formation of sequences which are identical except for the order in which the basic elements are chosen.
(6) A modification, which is being tested, is to bias the program towards choosing decompositions that involve the original inputs, when possible. This tends to reduce the number of levels in the circuit.

It is felt that this set of rules may not be the best possible, and investigations are being continued on other possibilities.

Results

One of the test cases that we have used is an eight-variable function quoted by Roth [1961]. This is shown in **Fig. 10. Fig. 11** shows a realisation found by Roth using an IBM 7090. On the KDF9 we have obtained a first implementation in 1·6 sec. with a cost of 32 and a final implementation in 68 sec. with a cost

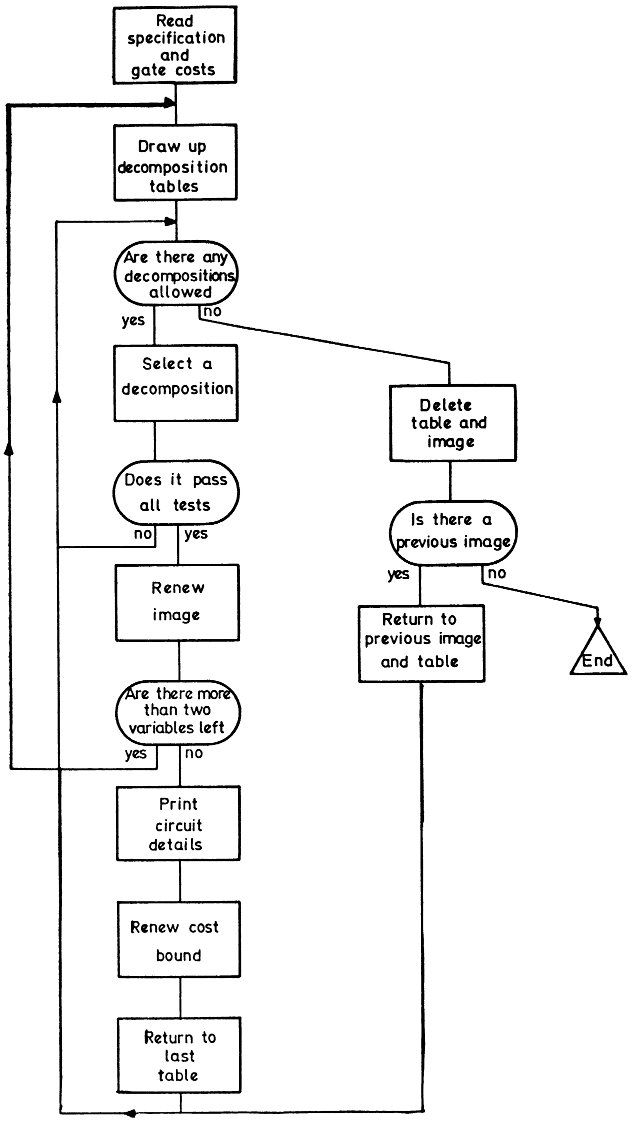

Read specification and gate costs
Draw up decomposition tables
Are there any decompositions allowed
yes
no
Select a decomposition
Delete table and image
Does it pass all tests
no
yes
Is there a previous image
yes
no
Renew image
Return to previous image and table
End
Are there more than two variables left
yes
no
Print circuit details
Renew cost bound
Return to last table

Fig. 9. The flow chart

ON-array	OFF-array	
A B C D E F G H	*A B C D E F G H*	*A B C D E F G H*
1 1 *x x x x x x*	0 1 *x x x x x* 0	0 *x x x x* 0 0 *x*
1 *x x* 1 *x x* 1 1	0 1 *x x x x* 0 *x*	0 *x x x x* 0 *x* 0
1 *x x* 0 1 1 *x x*	0 1 *x* 0 *x x x x*	*x* 0 *x* 1 *x x* 0 *x*
1 *x x x* 1 *x* 1 1	0 *x* 0 *x x x x x*	*x* 0 *x* 1 *x x x* 0
x x 1 1 *x x* 1 1	0 *x x* 0 0 *x x x*	*x* 0 *x* 0 0 *x x x*
x 0 1 0 1 1 *x x*	0 *x x* 1 *x x* 0 *x*	*x* 0 *x x* 0 *x* 0 *x*
x 0 1 *x* 1 *x* 1 1	0 *x x* 1 *x x x* 0	*x* 0 *x x* 0 *x x* 0
	0 *x x x* 0 *x* 0 *x*	*x* 0 *x x x* 0 0 *x*
	0 *x x x* 0 *x x* 0	*x* 0 *x x x* 0 *x* 0

Fig. 10. Specification quoted by Roth and used by us as a test case

of 26. These are shown in **Figs. 12** and **13.** The gate-cost for this run was the number of inputs. The additional gate-costs were tried:

2-input NOR gate : 4
1-input NOR gate : 1

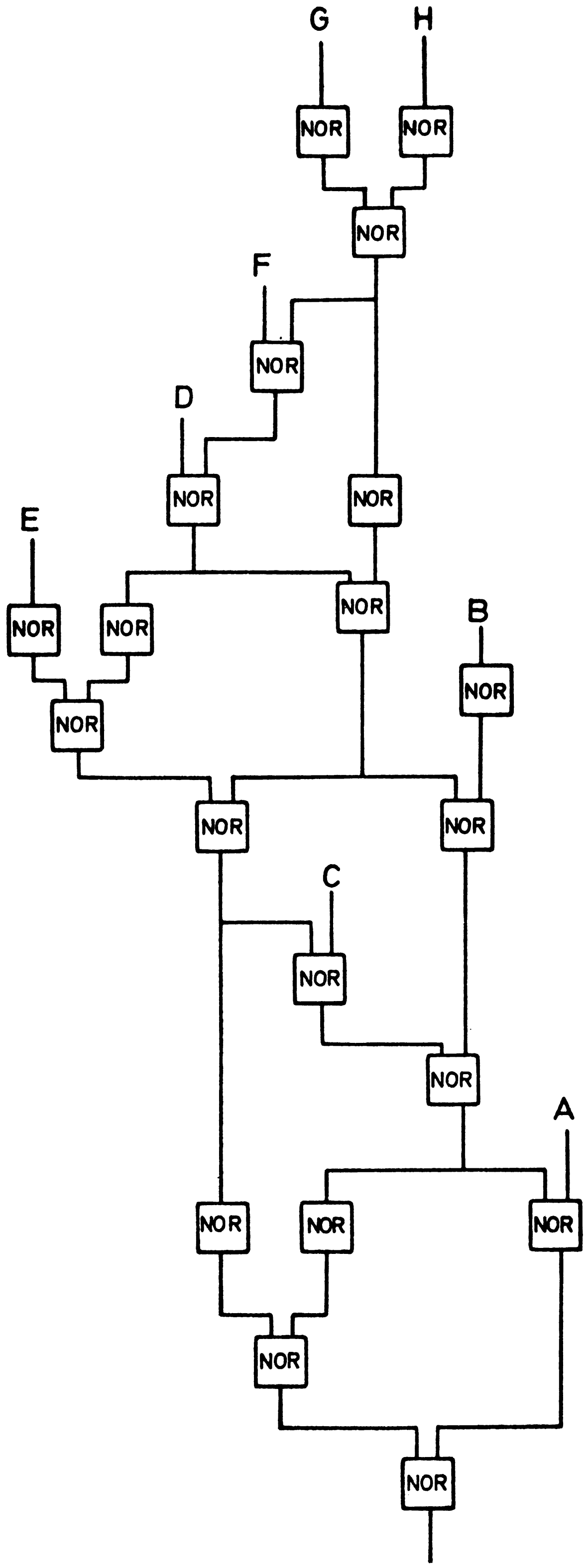


Fig. 12. First implementation on KDF9

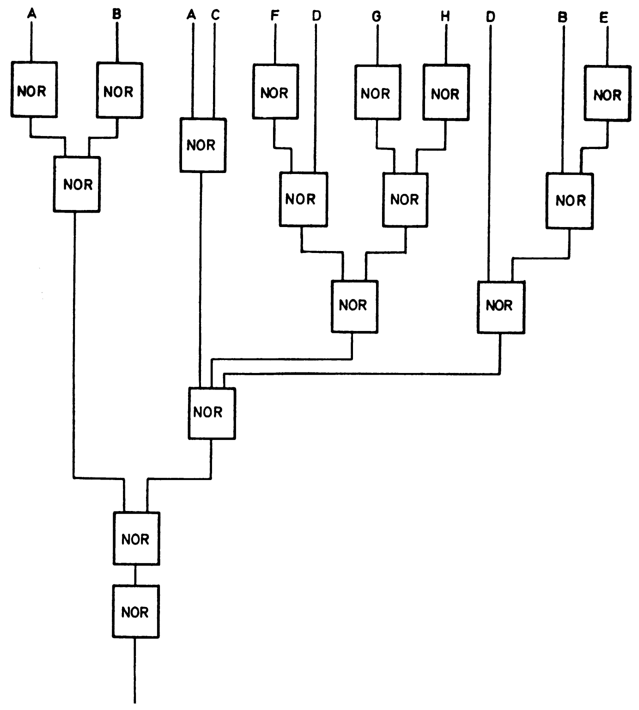


Fig. 11. Circuit found by Roth

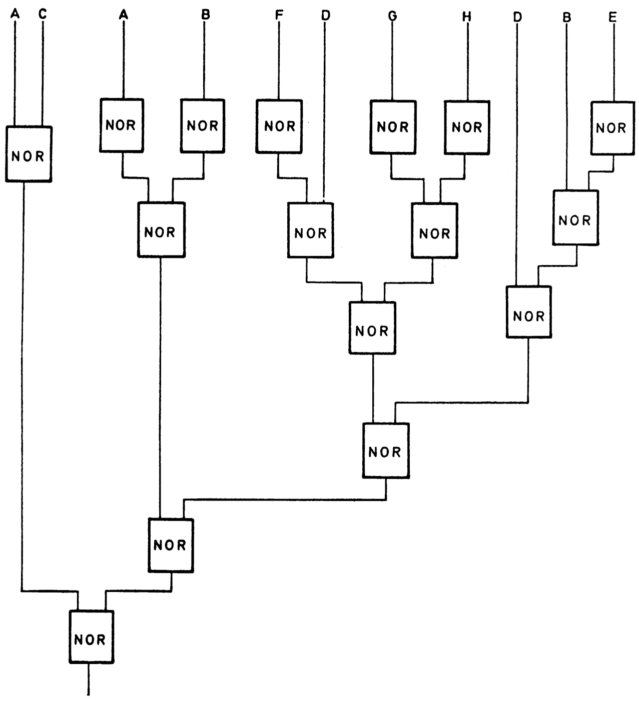


Fig. 13. Final implementation on KDF9

time for minimum circuit 52 sec.

2-input AND	: 2
2-input OR	: 2
NOT	: 1

time for minimum circuit 49 sec.

The value of the '1-step look ahead' procedure (rule 2 above) is shown by the result that without it the first circuit found had cost 47.

It will be noticed that Roth's solution contains a 3-input gate, but to what extent his program allows the choice of gates with more than two inputs we do not know. Certainly decomposition theory applied with vertex type gates can be used to give algorithms using gates with an arbitrary number of inputs. As Ashenhurst and Roth point out, one of the properties of vertex functions is that the existence of $\alpha(a, b, c)$ implies the existence of $\beta_1(a, b)$, $\beta_2(b, c)$ and $\beta_3(c, a)$ for some β_1, β_2, β_3, and vice versa. Thus, if it was desirable to extend the algorithm to 3- or 4-input gates this could be done still using our 2-input decomposition table, e.g. if the table shows that

$$\lambda_1 = \{a, b\} \quad V_{\beta_1} = 11$$
$$\lambda_2 = \{b, c\} \quad V_{\beta_2} = 10$$
$$\lambda_3 = \{a, c\} \quad V_{\beta_3} = 10$$

are possible then we can deduce the existence of $\lambda = \{a, b, c\}$ $V_\alpha = 110$.

Acknowledgements

This paper has been written as part of the research program for the Mechanical Engineering Research Laboratory of the English Electric Company Limited, Whetstone, Leicester. The paper is published with the permission of the English Electric Co., Ltd.

References

Roth, J. P., Karp, R. M., McFarlin, F. E., and Wilts, J. R. (1961). A Computer Program for the Synthesis of Combinational Switching Circuits, *Proceedings of the 2nd Annual Symposium on Switching Circuit Theory and Logical Design.* Published by A.I.E.E.

Roth, J. P., and Karp, R. M. (1962). Minimisation over Boolean Graphs, *IBM Journal*, Vol. 6, pp. 227–238.

Curtis, H. A. (1962). *Design of Switching Circuits*, D. Van Nostrand Co., Princeton.

Ashenhurst, R. L. (1959). The Decomposition of Switching Functions, Proceedings of an International Symposium on the Theory of Switching, April 2nd–5th, 1957. *The Annals of the Harvard Computation Laboratory*, XXIX, Harvard University Press, Cambridge, Mass., 1959, pp. 74–116.