Automated synthesis of digital multiplexer networks

A.E.A. Almaini, PhD, FIEE
J.F. Miller, PhD
L. Xu, BSc

Indexing terms: Combinational logic, Multiplexers, Universal logic modules, Cascade and tree networks

Abstract: A programmed algorithm is presented for the synthesis and optimisation of networks implemented with multiplexer universal logic modules. The algorithm attempts level by level optimisation selecting the control variables that result in minimum number of continuing branches. Cascaded networks, if realisable, are always found and given preference over tree networks, though mixtures of cascade and tree configurations are permitted. The algorithm is programmed in Fortran and tested for single and double control variable modules. In theory, the program can be used for any number of variables for completely and incompletely specified functions.

1 Introduction

The use of multiplexers as universal logic modules (ULMs) for the realisation of logic functions has attracted a great deal of attention during the last two decades following publications by Yau and Tang [1, 2]. Most of the effort was in finding minimal realisations for logic functions. This included linear programming and numerical methods [3], maximisation of the number of ones and zeros connected to the input [4], control variable selection with view to minimise the number of branches in ULM trees [5], as well as the use of decomposition and reduced dependence methods [6, 7].

A graphical method for cascade implementations based on Karnaugh maps was introduced by Tosser *et al.* [8]. The map method, however, limits the maximum number of variables that can be considered to six. An iterative method for cascade realisations using single control multiplexers was presented by Gorai *et al.* [9]. The method terminates if the function is not cascade realisable. This paper introduces a programmed algorithm which implements any logic function using minimal trees. A cascade connection, if possible, is considered a special form of a tree with a single input continuing into multilevel network. $M(c)$ is used to indicate a multiplexer ULM module with c control variable(s). This device has a single output and 2^c inputs. Any logic function can be implemented using ULMs connected as a tree with the first level (output stage) having a single ULM, the second level having a maximum of 2^c ULMs and so on. At any level, the logic function is expanded about the control variables used at that level using Shannon's expansion theorem [5]. The inputs may be connected to a constant (0 or 1), a variable $\dot{x}_i$, or a subfunction requiring further expansions and ULM levels. $\dot{x}_i$ is used to denote any variable x_i which can be true (x_i) or complemented ($\bar{x}_i$).

The algorithm works for any number of control variables though the program is only tested for $c = 1$ and $c = 2$, the most common cases. Because c is specified at each level, different size modules may be used at different levels if desired. The program automatically finds cascade connections, if available, as this reduces the number of continued branches. For tree type realisations, the algorithm permits mixed control variables within each level if this results in more branches being terminated with a constant or a single variable. The program can be used for any number of variables and can handle both completely and incompletely specified functions. The search for the best choice of control variables is exhaustive at the first level. At each subsequent level, an exhaustive search is carried out for the best choice of control variables amongst the remaining variables. The number of computations, however, decreases as the final level is approached. This approach was found to give reduced computation time though it does not guarantee global optimality in all cases.

2 Theory

Any n variable logic function $f(x_1, \ldots, x_n)$ can be expanded with respect to any $n - 1$ variables as follows

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{2^{n-1}-1} x_1^{i_1}, \ldots, x_{n-1}^{i_{n-1}} f(i_1, \ldots, i_{n-1}, x_n)$$

where the superscripts $i_1, \ldots, i_{n-1}$ form the binary representation of i

$$x_j^0 = \bar{x}_j$$
$$x_j^1 = x_j$$
$$j = 1, \ldots, n-1$$

The residue function $f(i_1, \ldots, i_{n-1}, x_n)$ is a function of one variable x_n which can assume any of the values x_n, $\bar{x}_n$, 0 or 1.

Such a function can be realised using a single multiplexer with $n - 1$ control inputs and 2^{n-1} data inputs [2].

If smaller multiplexer modules are used, a tree structure using l levels can be constructed to realise the function where $l_{max} = [(n-1)/c]$.

$[g]$ = smallest integer greater than or equal to g.

The tree may require up to $(I^l - 1)/(I - 1)$ modules where $I = 2^c$ [5].

By suitable selection of the control variables used at each level, the number of modules may be minimised, though the exchange of control variables for the same

Paper 8847E (C3), first received 14th May 1991 and in revised form 21st February 1992

The authors are with the Department of Electrical, Electronic & Computer Engineering, Napier University, 219 Colinton Road, Edinburgh EH14 1DJ, United Kingdom

module will only permute the connections to that module and does not result in any saving.

If $2^c - 1$ of the inputs terminate with a variable $\dot{x}_i$ or a logical constant and only one input continues into the next level, a cascade is generated where a single module is used in each level.

In practice, however, functions that can be implemented with a simple cascade, or require a complete tree are rare. More often, functions lie somewhere in between where some inputs may not be connected to other modules while other inputs connect to modules in the next level and therefore form an incomplete tree as will be illustrated in the examples in Section 3.

It is the aim, therefore, to identify control variables that eliminate as many branches as possible, and reduce the number of levels and modules required without having to resort to completely exhaustive search.

Theorem 2.1: For an n variable function realised by multiplexer tree; if a c control module at level l has an input

(i) 0

there are no entries in the minterm table corresponding to this input, or

(ii) 1

then there are 2^{n-cl} entries.

Proof:

(i) If a module has a 0 data input, this input cannot be selected, thus there are no entries in the minterm table corresponding to the control selection.

(ii) If an input is 1, all unselected variable can take any value. There are $n - cl$ variables unselected (those not required to select the 1 input) therefore there must be 2^{n-cl} minterm entries.

Theorem 2.2: For an n variable function realised by ULM tree, if a c control module at level l has input $\dot{x}_j$ (where j is not a control used in the $l - 1$ levels preceding the module in question) there are 2^{n-cl-1} entries in the minterm table (with $\dot{x}_j$ fixed either as x_j or $\bar{x}_j$).

Proof: To select the input in question, lc variables are required as controls. Thus if $\dot{x}_j$ is input there must be n-$(lc + 1)$ variables unspecified. These can have any value. Thus there must be 2^{n-cl-1} entries in the minterm table. Note $\dot{x}_j$ is fixed so all entries must correspond to either $\dot{x}_j = \bar{x}_j$ or $\dot{x}_j = x_j$.

Lemma 2.1: Connected ULMs from different levels of a network cannot share a common control variable [9].

Theorem 2.3: A $M(c)$ multiplexer can realise any $c + 1$ variable function [10].

These theorems are utilised in Steps 3 and 4 of the minimisation algorithm in the following Section.

3 Algorithm

Step 1: Get the minterms of the given function. Set the level $l = 1$. Calculate the number of variables n.

Step 2: Get the number of control variables per module c and check whether the number of variables prior to level l, $n - c(l - 1) \leqslant c + 1$. If so, the tree can finish with any choice of remaining variables. If not, continue.

Step 3: Check if there is any c-tuple of variables $\{x_{i_r}: r = 1, \ldots, c\}$ for which the number of simultaneous occurences in the minterm table

$$S_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c}) = 0 \text{ or} \tag{3a}$$

$$S_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c}) = 2^{n-cl} \tag{3b}$$

If either of eqn. 3a or eqn. 3b is true, increment the saved branch counter $y_c(x_{i_1}, \ldots, x_{i_c})$. Keep a record of the input $I_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c}) = 0$ or 1 respectively.

Step 4: Check if there is any $c + 1$ tuple of variables which satisfy

$$S_{c+1}(\dot{x}_{i_1}, \ldots, \dot{x}_{i_{c+1}}) = 2^{n-cl-1}. \tag{4a}$$

If so, check that there is some variable x_{i_k}, say, which, for all simultaneous occurrences, has the same value

$$S_{c+1}(\dot{x}_{i_1}, \ldots, \overline{\dot{x}_{i_k}}, \ldots, \dot{x}_{i_{c+1}}) = 0 \tag{4b}$$

If so, increment the branch saved counter

$$y_c(x_{i_1}, \ldots, x_{i_{k-1}}, x_{i_{k+1}}, \ldots, x_{i_{c+1}})$$

Keep a record of the input

$$I_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_{k-1}}, \dot{x}_{i_{k+1}}, \ldots, \dot{x}_{i_{c+1}}) = \dot{x}_{i_k}$$

Step 5: According to the results of Steps 3 and 4, calculate which c tuple of variables (to be chosen as controls to module) $x_{h_1}, \ldots, x_{h_c}$ maximises the number of saved branches

$$Y_c(x_{h_1}, \ldots, x_{h_c}) = \max \{y_c(x_{i_1}, \ldots, x_{i_c})\}$$

Step 6: According to the choice of Step 5, there are $2^c - Y_c$ inputs to the module in question whose input is neither a constant nor a single variable.

Obtain the reduced subfunctions for the inputs separately and derive their minterm tables. If any subfunctions are identical, inputs can be connected together and further branches saved. Increment the level $l := l + 1$. For each unique subfunction, go to Step 2 and repeat the procedure.

Example 1: Implement the following function using $M(2)$

$$f = \sum (4, 5, 6, 7, 8, 10, 11, 12, 14, 15, 20, 21, 22, 23, 27, 28, 29, 30, 32, 33, 34, 35, 38, 39, 42, 43, 46, 47, 48, 49, 50, 51, 54, 55, 58, 59, 62, 63)$$

Step 1: $n = 6, l = 1$

Step 2: $c = 2, n - c(l - 1) = 6 > c + 1 = 3$. The minterm table is given in Table 1.

Steps 3 and 4 are carried out for all possible control pairs $x_i, x_j, i \neq j$ and, for each pair, determine how many inputs have either variables coming in or fixed inputs 0 or 1. For level $l = 1$, $2^{n-cl} = 2^4 = 16$, $2^{n-cl-1} = 8$. Thus it is necessary to find simultaneous pairs $S_2(\dot{x}_i, \dot{x}_j) = 0$, 16 and simultaneous triples $S_3(\dot{x}_i, \dot{x}_j, \dot{x}_k) = 8$. The latter being subject to eqn. 4b.

Control pair x_1x_2

$S_2(\bar{x}_1, \bar{x}_2) = 10$. This branch cannot be saved.

$S_2(\bar{x}_1, x_2) = 8$. Eqn. 4a is potentially satisfied

but

$S_3(\bar{x}_1, x_2, x_j) \neq 8$. $\forall j$ $(j \neq 1, 2)$. This branch cannot be saved.

$S_2(x_1, \bar{x}_2) = 10$. This branch cannot be saved.

$S_2(x_1, x_2) = 10$. This branch cannot be saved.

No branches can be saved at the first level, if x_1 and x_2 are chosen.

Control pair x_1, x_3

$S_2(\bar{x}_1, \bar{x}_3) = 8$. Eqn. 4$a$ is potentially satisfied.

Table 1: Minterms for Example 1

x_1	x_2	x_3	x_4	x_5	x_6
0	0	0	1	0	0
0	0	0	1	0	1
0	0	0	1	1	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	0
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	0	1	1	1	1
0	1	0	1	0	0
0	1	0	1	0	1
0	1	0	1	1	0
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	0	0
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	0	1	1
1	0	0	1	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	0	1	1
1	0	1	1	1	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	0	1	1
1	1	0	1	1	0
1	1	0	1	1	1
1	1	1	0	1	0
1	1	1	0	1	1
1	1	1	1	1	0
1	1	1	1	1	1

$S_3(\bar{x}_1, \bar{x}_3, x_4) = 8$, and $S_3(\bar{x}_1, \bar{x}_3, \bar{x}_4) = 0$. This branch can be saved.
$I_2(\bar{x}_1, \bar{x}_3) = x_4$.
$y_2(x_1, x_3) = 1$.
For the next inputs
$S_2(\bar{x}_1, x_3) = 10$. $S_2(x_1, \bar{x}_3) = 12$. These branches cannot be saved.
$S_2(x_1, x_3) = 8$. Eqn. 4*a* is potentially satisfied.
$S_3(x_1, x_3, x_5) = 8$, and $S_3(x_1, x_3, \bar{x}_5) = 0$. Another branch is saved.
$I_2(x_1, x_3) = x_5$.
$y_2(x_1, x_3) = 2$.

Carrying out the tests in Steps 3 and 4 for all control pairs causes the conclusion that the most saved branches for any pair is two. Choose x_1 and x_3 as the initial control variables (Step 5). We depict the initial module in Fig. 1.

Step 6: Obtain the reduced subfunctions for the non-terminating inputs $\bar{x}_1 x_3$ and $x_1 \bar{x}_3$.

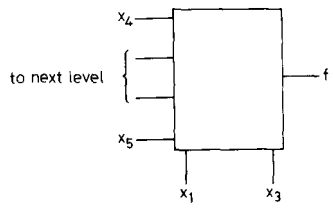

Fig. 1 *First stage realisation of example 1*

Increment the level $l = 2$. $2^{n-cl} = 4$, $2^{n-cl-1} = 2$. Go to Step 2. $n - c(l-1) = 4 > c + 1 = 3$.
For input $\bar{x}_1 x_3$ (Table 2) carry out Steps 3, 4 and 5.
Control pair x_2, x_5
$S_2(\bar{x}_2, \bar{x}_5) = 2$. Eqn. 4*a* is satisfied as $S_3(\bar{x}_2, \bar{x}_5, \bar{x}_6) = 2$. Eqn. 4*b* is satisfied.
$I_2(\bar{x}_2, \bar{x}_5) = \bar{x}_6$, $y_2(x_2, x_5) = 1$.
$S_2(\bar{x}_2, x_5) = 4$. Eqn. 3*b* is satisfied. $I_2(\bar{x}_2, x_5) = 1$, $y_2(x_2, x_5) = 2$

Table 2: Reduced minterm table for input $\bar{x}_1 x_3$

x_2	x_4	x_5	x_6
0	0	0	0
0	0	1	0
0	0	1	1
0	1	0	0
0	1	1	0
0	1	1	1
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0

$S_2(x_2, \bar{x}_5) = 2$. Eqn. 4*a* is satisfied as $S_3(x_2, \bar{x}_5, x_4) = 2$. Eqn. 4*b* is satisfied.
$I_2(x_2, \bar{x}_5) = x_4$, $y_2(x_2, x_5) = 3$.
$S_2(x_2, x_5) = 2$. $S_3(x_2, x_5, \dot{x}_j) \neq 2$, $j = 4$ or 6. Eqn. 4*a* is not satisified. Thus control pair x_2, x_5 saves three branches.

On considering all other pairs of controls, it is found that no pair saves more than three branches, hence x_2 and x_5 are the chosen pair.

For input $x_1 \bar{x}_3$ it is found that, for control pair $x_2 x_4$, all inputs are terminating so $x_2 x_4$ are the chosen controls and no more tests are needed.

Step 6: There is only one reduced subfunction, corresponding to input $\bar{x}_1 x_3 x_2 x_5$ as given in Table 3.

Table 3: Reduced minterm table for input $\bar{x}_1 x_3 x_2 x_5$

x_4	x_6
0	1
1	0

Increment the level $l = 3$. Go to step 2. $n - c(l-1) = 2 < c + 1 = 3$. Choose any pair of control variables. Choose $x_4 x_6$. Thus the complete tree is given in Fig. 2.

Modification for incompletely specified functions
In this Section a dash indicates quantities determined from the minterm table of incompletely specified terms.

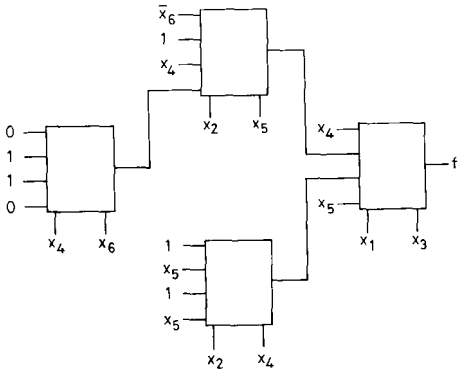

Fig. 2 *Complete implementation of example 1*

Step 3′: If neither eqn. 3*a* nor eqn. 3*b* are true, check whether

$$S_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c}) + S'_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c}) = 2^{n-cl} \quad (3)'$$

If so, $I_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c}) = 1$. Increment the saved branch counter

$$y_c(x_{i_1}, \ldots, x_{i_c})$$

Step 4′: If eqn. 4*a* is not satisfied and

$$S_{c+1}(\dot{x}_{i_1}, \ldots, \dot{x}_{i_{c+1}}) < 2^{n-cl-1} \quad (4a)'$$

Check whether there is a variable x_{i_k} which satisfies

$$S_{c+1}(\dot{x}_{i_1}, \ldots, \bar{\dot{x}}_{i_k}, \ldots, \dot{x}_{i_{c+1}}) = 0 \quad (4b)'$$

$$S_{c+1}(\dot{x}_{i_1}, \ldots, \dot{x}_{i_k}, \ldots, \dot{x}_{i_{c+1}}) + S'_{c+1}(\dot{x}_{i_1}, \ldots, \dot{x}_{i_k}, \ldots, \dot{x}_{i_{c+1}}) = 2^{n-cl-1} \quad (4c)'$$

If so, input

$$I_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_{k-1}}, \dot{x}_{i_{k+1}}, \ldots, \dot{x}_{i_{c+1}}) = \dot{x}_{i_k}$$

Increment the branch saved counter

$$y_c(x_{i_1}, \ldots, x_{i_{k-1}}, x_{i_{k+1}}, \ldots, x_{i_{c+1}})$$

Example 2: Implement the following incompletely specified function using $M(2)$.

$$f = \sum(1, 2, 3, 4, 5, 6, 8, 16, 17, 18, 21, 22, 29, 31) + \sum_d(7, 10, 12, 13, 19, 24, 25, 26, 27)$$

Step 1: $n = 5, l = 1$

Step 2: $c = 2$, $n - c(l-1) = 5 > c + 1$. The minterm table is given in Table 4.

Table 4: Minterm table for example 2 including don't care conditions

	x_1	x_2	x_3	x_4	x_5
	0	0	0	0	1
	0	0	0	1	0
	0	0	0	1	1
	0	0	1	0	0
	0	0	1	0	1
	0	0	1	1	0
	0	1	0	0	0
	1	0	0	0	0
	1	0	0	0	1
	1	0	0	1	0
	1	0	1	0	1
	1	0	1	1	0
	1	1	1	0	1
	1	1	1	1	1
d_1	0	0	1	1	1
d_2	0	1	0	1	0
d_3	0	1	1	0	0
d_4	0	1	1	0	1
d_5	1	0	0	1	1
d_6	1	1	0	0	0
d_7	1	1	0	0	1
d_8	1	1	0	1	0
d_9	1	1	0	1	1

Steps 3′ and 4′: One exhaustively searches all pairs of variables x_i, $x_j i \neq j$ and for each pair determine how many inputs have either variables coming in or fixed inputs 0 or 1. To do this, apply the tests described in Steps 3′ and 4′. For the first level $2^{n-cl} = 8$ and $2^{n-cl-1} = 4$. Thus one is looking for simultaneous pairs $S_2(\dot{x}_i, \dot{x}_j) = 0$ or $S_2(\dot{x}_i, \dot{x}_j) + S'_2(\dot{x}_i, \dot{x}_j) = 8$ and simultaneous triples $S_3(\dot{x}_i, \dot{x}_j, \dot{x}_k) + S'_3(\dot{x}_i, \dot{x}_j, \dot{x}_k) = 4$.

Control pair x_1, x_2

$S_2(\bar{x}_1, \bar{x}_2) = 6$, $S'_2(\bar{x}_1, \bar{x}_2) = 1$ (d_1). Eqn. 3′ and eqn. 4a′ are not satisfied.

$S_2(\bar{x}_1, x_2) = 1$, $S'_2(\bar{x}_1, x_2) = 3$ (d_2, d_3, d_4). However eqn. 4*c*′ cannot be satisfied for any triple.

$S_2(x_1, \bar{x}_2) = 5$, $S'_2(x_1, \bar{x}_2) = 1$ (d_5). Eqn. 3′ and eqn. 4*a*′ are not satisfied.

$S_2(x_1, x_2) = 2$, $S'_2(x_1, x_2) = 4$ (d_6, d_7, d_8, d_9). Eqn. 3′ is not satisfied.

It is found that $S_3(x_1, x_2, \bar{x}_5) = 0$ and $S_3(x_1, x_2, x_5) + S'_3(x_1, x_2, x_5) = 4$. Eqn. 4*b*′ and eqn. 4*c*′ are satisified provided we choose d_7 and d_9. Thus $I_c(x_1, x_2) = x_5$. Increment the saved branch counter $y_2(x_1, x_2) = 1$. Now select another control pair.

Control pair x_1, x_3

$S_2(\bar{x}_1, \bar{x}_3) = 4$, $S'_2(\bar{x}_1, \bar{x}_3) = 1$ (d_2). Eqn. 3′ is not satisfied. Eqn. 4*c*′ is potentially satisfied. However, examination of the minterm table shows $S_3(\bar{x}_1, \bar{x}_3, \dot{x}_j) \neq 4\ \forall j, j \neq 1, 3$. So this input cannot be saved. $S_2(\bar{x}_1, x_3) = 3$, $S'_2(\bar{x}_1, x_3) = 3$ (d_1, d_3, d_4). Eqn. 3′ is not satisfied. Eqn. 4*c*′ is potentially satisfied. Choosing don't care d_1, we find $S_3(\bar{x}_1, x_3, x_2) = 0$ and $S_3(\bar{x}_1, x_3, \bar{x}_2) + S'_3(\bar{x}_1, x_3, \bar{x}_2) = 4$ so that $I_2(\bar{x}_1, x_3) = \bar{x}_2$ and $y_2(x_1, x_3) = 1$.

$S_2(x_1, \bar{x}_3) = 3$, $S'_2(x_1, \bar{x}_3) = 5$ $(d_5 - d_9)$. Potentially eqn. 3′ or eqn. 4*c*′ are satisfied, as $S_2(x_1, \bar{x}_3) + S'_2(x_1, \bar{x}_3) = 8$ giving $I_2(x_1, \bar{x}_3) = 1$.

$S_3(x_1, \bar{x}_3, x_2) = 0$ and $S_3(x_1, \bar{x}_3, \bar{x}_2) + S'_3(x_1, \bar{x}_3, \bar{x}_2) = 4$ (d_5) giving $I_2(x_1, \bar{x}_2) = \bar{x}_2$.

Thus again an input can be saved, $y_2(x_1, x_3) = 2$ and there is a choice of input. Considering the last input, it is found that $S_2(x_1, x_3) = 2$.

$S'_2(x_1, x_3) = 0$. Neither eqn. 3′ or eqn. 4′ can be satisfied. Continuing to work through all possible control pairs, it is found that control choice x_2 and x_5 also saves two branches, all other control choices save less than two branches.

Step 5: Control variables x_1 and x_3 are chosen for the first level. The first module is shown in Fig. 3.

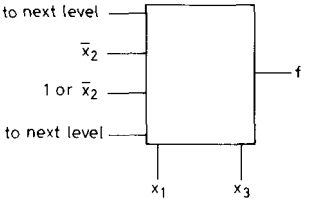

Fig. 3 *First stage implementation of example 2*

Step 6: On obtaining the reduced subfunction for the nonterminating inputs $\bar{x}_1\bar{x}_3$ and x_1x_3, it is found that the reduced subfunctions are nonidentical, so no further branches can be saved.

Increment the level, $l = 2$. For each subfunction we go to Step 2. It is found that $n - c(l-1) = 3 = c + 1$, hence the tree network can finish with control choices x_2, x_4. The complete tree realisation is seen in Fig. 4.

4 Cascade realisable functions

The algorithm performs exhaustive search at the first level (stage). The control variables resulting in the most non-continuing branches (branches terminating with $\dot{x}_i$, 0 or 1) are selected. No further levels are required if all the inputs terminate. If choice is possible, then any can be selected with the knowledge that the cascade solution, if

any, will always be found. That this statement is true can be seen from Theorem 4.1. To illustrate this situation consider the following example.

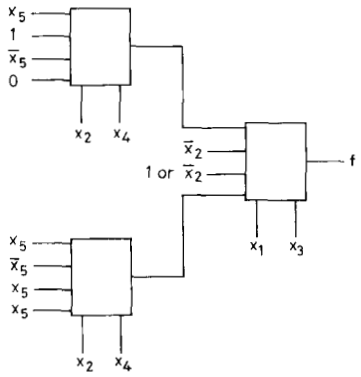

Fig. 4 *Complete implementation of example 2*

Example 3

$f = \sum (0, 1, 4, 9, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 34, 35, 38, 39, 42, 43, 46, 47, 48, 49, 52, 53, 56, 57, 60, 61)$

The computer program gives the solution in Fig. 5.

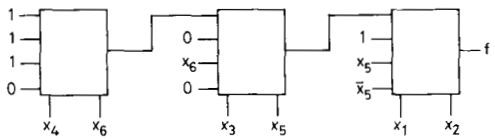

Fig. 5 *Cascade implementation of example 3*

In fact, x_1x_2 is not the only choice to make the first level cascade realisable. x_1x_5 and x_2x_5 are other choices. Because the program will keep the first one it has found, the question arises as to whether it can happen that some choices may give cascade networks and others do not.

The Karnaugh map of Example 3 (Fig. 6) is given to illustrate that, if there is choice of cascade realisability at a certain level, the algorithm could find the cascade network at the next level if it exists *irrespective* of the control pair choice.

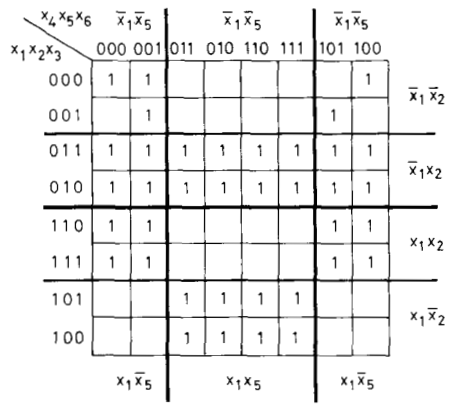

Fig. 6 *Split K-map of example 3*

Fig. 6 shows the *K*-map being split with respect to x_1x_2 or x_1x_5. For pair x_1x_2, all parts except $\bar{x}_1\bar{x}_2$ are typical loops [8] which guarantees that their inputs are connected to constants or single variable rather than other modules. For pair x_1x_5, the $\bar{x}_1\bar{x}_5$ part is the only region not containing a typical loop. Under the condition that both choices are cascade realisable at the first level, all parts except where $\bar{x}_1\bar{x}_2$ and $\bar{x}_1\bar{x}_5$ intersect must be typical loops. The subfunction of the next level is defined by the minterms in the regions $\bar{x}_1\bar{x}_2$ or $\bar{x}_1\bar{x}_5$ depending on which choice has been made. However, in either case, the minterms which are included in the overlapping area of $\bar{x}_1\bar{x}_2$ and $\bar{x}_1\bar{x}_5$ determine whether the remaining subfunction is cascade realisable at the next level. The above argument is now generalised with the aid of Theorem 4.1.

Theorem 4.1: Let f be an $M(c)$ $(c = 1, 2)$ cascade realisable function at level l with controls $x_n(c = 1)$ or x_n, x_m $(c = 2)$ and at level $l + 1$ with controls $x_j(c = 1)$ or x_j, $x_k(c = 2)$. In addition, let f be level l cascade realisable with different controls x_r $(c = 1)$ or x_r, x_s $(c = 2)$ then *controls can always be found for which* f_{l+1} *is cascade realisable.*

Proof: Consider the case of two control variables $(c = 2)$. This is depicted in Fig. 7*a*

$$f_l = \bar{x}_n\bar{x}_m f_{l+1} + a_1\bar{x}_n x_m + a_2 x_n\bar{x}_m + a_3 x_n x_m \quad (1a)$$

$$f_{l+1} = \bar{x}_j\bar{x}_k f_{l+2} + b_1\bar{x}_j x_k + b_2 x_j\bar{x}_k + b_3 x_j x_k \quad (1b)$$

$$f_{l+2} = f_{l+2}(x_i) \quad i \neq n, m, j, k \quad (1c)$$

From the theorem conditions f_l is also given by (see Fig. 7*b*)

$$f_l = \bar{x}_r\bar{x}_s f''_{l+1} + c_1\bar{x}_r x_s + c_2 x_r\bar{x}_s + c_3 x_r x_s \quad (1d)$$

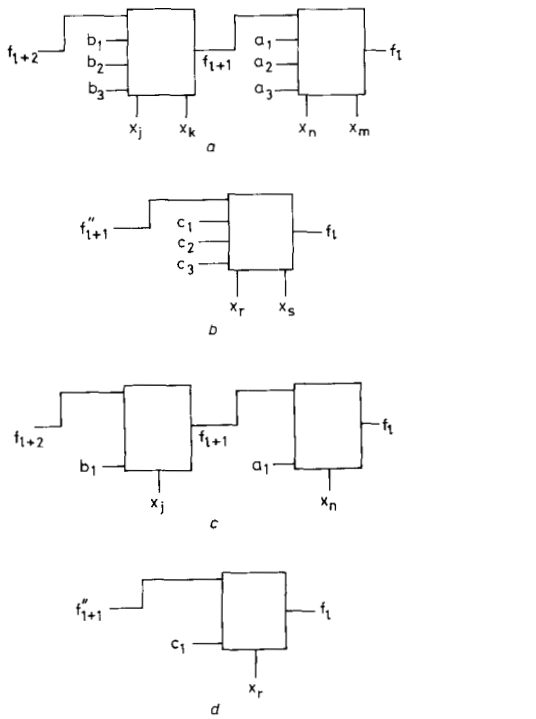

Fig. 7 *Alternative cascade realisations of logic function*

Equating eqn. 1*a* and eqn. 1*d* and multiplying (AND operation) by $\bar{x}_r \bar{x}_s$ we find

$$f_l = \bar{x}_r \bar{x}_s f'_{l+1} + c_1 \bar{x}_r x_s + c_2 x_r \bar{x}_s + c_3 x_r x_s \quad (2a)$$

$$f'_{l+1} = \bar{x}_n \bar{x}_m f'_{l+2} + a_1 \bar{x}_n x_m + a_2 x_n \bar{x}_m + a_3 x_n x_m \quad (2b)$$

$$f'_{l+2} = \bar{x}_j \bar{x}_k f'_{l+3} + b_1 \bar{x}_j x_k + b_2 x_j \bar{x}_k + b_3 x_j x_k \quad (2c)$$

$$f'_{l+3} = f'_{l+3}(x_i) = f_{l+2}(x_i) \quad i \neq r, s, n, m, j, k \quad (2d)$$

If any b_i or a_i involve x_r or x_s because of multiplication by $\bar{x}_r \bar{x}_s$ in eqn. 2*a* these terms will vanish. While, if $\bar{x}_r$ or $\bar{x}_s$ is involved, the inputs become 1.

If $x_n = x_r$ a control is shared. In this case eqn. 2*b* becomes

$$f'_{l+1} = \bar{x}_m f'_{l+2} + a_1 x_m \quad (3)$$

If, in addition, x_m is chosen to equal x_j then eqn. 3, eqn. 2*b* and eqn. 2*c* can be written as follows

$$f'_{l+1} = \bar{x}_j \bar{x}_k f'_{l+2} + b_1 \bar{x}_j x_k + a_1 x_j \bar{x}_k + a_1 x_j x_k \quad (4a)$$

$$f'_{l+2} = f'_{l+3} \quad (4b)$$

Thus it can be seen that f'_{l+1} is cascade realisable with controls j, k. If $x_j = x_r$, eqn. 2*a* and eqn. 2*b* are unchanged so again the theorem is true. If $x_n = x_r$ and $x_k = x_s$, eqn. 2*a* is unchanged and eqn. 2*b* becomes eqn. 3 while eqn. 2*c* becomes

$$f'_{l+2} = \bar{x}_j f'_{l+3} + b_2 x_j \quad (5)$$

Eqn. 3 and eqn. 5 combine to give

$$f'_{l+1} = \bar{x}_m \bar{x}_j f'_{l+2} + b_2 \bar{x}_m x_j + a_1 x_m \bar{x}_j + a_1 x_m x_j$$

Hence here too f'_{l+1} is cascade realisable.

For a single control ($c = 1$) eqns. 1*a–d* become (see Figs. 7*c* and 7*d*

$$f_l = \bar{x}_n f_{l+1} + a_1 x_n \quad (6a)$$

$$f_{l+1} = \bar{x}_j f_{l+2} + b_1 x_j \quad (6b)$$

$$f_{l+2} = f_{l+2}(x_i) \quad i \neq n, j \quad (6c)$$

$$f_l = \bar{x}_r f'_{l+1} + c_1 x_r \quad (6d)$$

As before, equating eqn. 6*a* and eqn. 6*d* and multiplying by $\bar{x}_r$

$$f_l = \bar{x}_r f'_{l+1} + c_1 x_r \quad (7a)$$

$$f'_{l+1} = \bar{x}_n f'_{l+2} + a_1 x_n \quad (7b)$$

$$f'_{l+2} = \bar{x}_j f'_{l+3} + b_1 x_j \quad (7c)$$

$$f'_{l+3} = f_{l+2}(x_i) \quad i \neq n, j, r \quad (7d)$$

It is clear from the above equations that f'_{l+1} is cascade realisable for any value of r (including j). Hence the Theorem is proved.

5 Conclusion

A programmed algorithm for the synthesis of optimised multiplexer networks is presented. Level by level optimisation techniques were employed and found to give optimal results in all examples attempted. Multilevel optimisation techniques may further improve the results, but would significantly increase the complexity of the procedure and hence increase the computation time. The algorithm can handle any number of variables for completely and incompletely specified logic functions. Redundant variables, if any, are automatically eliminated. The program was written in Fortran-77 and run on a DEC micro VAX II. Computation time was found to increase with the number of variables, number of minterms and number of control variables. All the examples given in this paper took less than 8 seconds of CPU time. A larger example of 20 variables and 200 minterms requiring 521 $M(2)$ modules, took about 30 minutes of CPU time. For functions much larger than 20 variables, heuristic based algorithms may be desirable. The program listing can be obtained from one of the authors.

6 References

1 YAU, S.S., and TANG, C.K.: 'Universal logic circuits and their modular realisation', *AFIPS Conf. Proc.*, 1968, **32**, pp. 297–305

2 YAU, S.S., and TANG, C.K.: 'Universal logic modules and their applications', *IEEE Trans.*, 1970, **C-19**, pp. 141–149

3 TABLOSKI, T.F. Jr.: 'Analysis and synthesis of minimal multiplexer universal logic module trees'. PhD thesis, 1973, Purdue University

4 WHITEHEAD, D.G.: 'Algorithm for logic circuit synthesis using multiplexers', *Electr. Lett.*, 1977, **13**, pp. 355–356

5 ALMAINI, A.E.A., and WOODWARD, M.E.: 'An approach to the control variable selection problem for universal logic modules', *Dig. Proc.*, 1977, **3**, pp. 189–206

6 LANGDON, G.G. Jr.: 'A decomposition chart technique to aid in realisations with multiplexers', *IEEE Trans.*, 1978, **C-27**, pp. 154–159

7 ALMAINI, A.E.A.: 'Sequential machine implementation using ULMs', *IEEE Trans.*, 1978, **C–27**, pp. 951–960

8 TOSSER, A.J., and AOULAD SYAD, D.: 'Cascade networks of logic functions built in multiplexer units', *IEE Proc. E*, 1980, **127**, (2), pp. 64–68

9 GORAI, R.K., and PAL, A.: 'Automated synthesis of combinational circuits by cascade networks of multiplexers', *IEE Proc. E*, 1990, **137**, (2), pp. 164–170

10 PAL, A.: 'An algorithm for optimal logic design using multiplexers', *IEEE Trans.*, 1986, **C–35**, pp. 755–757