Reed–Muller universal logic module networks

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

Indexing terms: Combinational logic, Optimised networks, Reed–Muller expansions, Universal logic modules

Abstract: The paper describes Reed–Muller universal logic modules (RM–ULMs) and their use for the implementation of logic functions given in Reed–Muller (RM) form. A programmed algorithm is presented for the synthesis and optimisation of RM–ULM networks. The level-by-level minimisation procedure is based on the selection of control variables at different levels with the aim of maximising the number of discontinued branches and hence minimising the number of modules required to implement a given function. The algorithm is programmed in Fortran and can be used to realise fixed-polarity generalised Reed–Muller (GRM) expansions of any polarity and any number of variables.

List of symbols

$\dot{x}_i$ = x_i or $\bar{x}_i$
$\underline{x}_i$ = x_i is absent
$\hat{x}_i$ = $\dot{x}_i$ or $\underline{x}_i$
$S_c(\dot{x}_1, \ldots, \dot{x}_c)$ = number of simultaneous occurrences of $\dot{x}_1, \ldots, \dot{x}_c$ in the piterm table
$I_c(\dot{x}_1, \ldots, \dot{x}_c)$ = the $\dot{x}_1, \ldots, \dot{x}_c$ data input in a c-control variable module
f_i = GRM function with polarity i
$f(i)$ = output of RM–ULM(i)
p_i = ith piterm
$p_i(k)$ = kth bit of the ith piterm
y = number of saved branches
Y = max (y)

1 Introduction

The use of multiplexers as universal logic modules (ULMs) for the implementation of combinational logic is well documented [1–4]. A ULM of a given number of variables can be used to realise any Boolean logic function of up to that number of variables. ULMs can be connected as a network to realise functions containing large numbers of variables. Various methods have been developed for the optimisation of the resulting ULM networks [2–4].

Logic functions can also be expressed in terms of RM expansions (polynomials). RM expansions have certain properties, such as ease of complementing and testing, that make them attractive in some applications [5]. Literature has been published recently discussing design and minimisation techniques for RM logic, derivation of various polarities, as well as conversion between RM and Boolean forms [6–9].

For the implementation of RM functions, RM–ULMs can be used as individual modules or as networks. Map methods may be used for the synthesis of functions with up to six variables [9], but these become impractical for the realisation of economical networks for larger functions. In this paper, a programmed algorithm is described for the synthesis and optimisation of RM–ULM networks which can be used for any number of variables.

2 Reed–Muller universal logic module

A RM–ULM with c-control (selection) inputs, is a device with 2^c data inputs and a single output, abbreviated from now on as RM–ULM(c). The symbol of the RM–ULM(c) is given in Fig. 1. The output of the device is as follows

$$f(c) = g_0 \oplus g_1\dot{x}_1 \oplus g_2\dot{x}_2 \oplus g_3\dot{x}_2\dot{x}_1 \oplus \cdots \oplus g_{2^c-1}\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \quad (1)$$

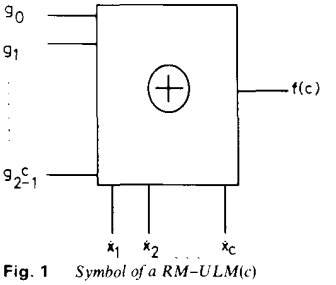

Fig. 1 *Symbol of a RM–ULM(c)*

For example

$$f(1) = g_0 \oplus g_1\dot{x}_1$$

and

$$f(2) = g_0 \oplus g_1\dot{x}_1 \oplus g_2\dot{x}_2 \oplus g_3\dot{x}_2\dot{x}_1$$

These are the two modules that shall be used in this paper.

Theorem 1: A RM–ULM(c) can directly synthesise any GRM expansion of c variables by employing the function domain coefficients as data inputs and function variables as control inputs of the module.

L. McKenzie acknowledges the support of the Science and Engineering Research Council under Grant GR/F 74288.

Paper 9337E (C3), first received 29th June and in revised form 7th October 1992

The authors are with the Electrical Electronic and Computer Engineering Department, Napier University, Edinburgh EH14 1DJ, Scotland, United Kingdom

Proof: A GRM expansion of c variables:

$$\begin{aligned} f(x_c, x_{c-1}, \ldots, x_1) &= b_0 \oplus b_1\dot{x}_1 \oplus b_2\dot{x}_2 \oplus b_3\dot{x}_2\dot{x}_1 \\ &\quad \oplus \cdots \oplus b_{2^c-2}\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_2 \\ &\quad \oplus b_{2^c-1}\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \end{aligned} \tag{2}$$

$$b_j \in [0, 1] \quad j = 0, 1, \ldots, 2^c - 1$$

Note that eqn. 2 is identical to eqn. 1 when b_j is equated to g_j, thus proving the theorem.

Theorem 2: A RM–ULM(c) can implement any GRM expansion of $c + 1$ variables.

Proof: A GRM expansion of $c + 1$ variables:

$$\begin{aligned} f(x_{c+1}, x_c, \ldots, x_1) &= b_0 \oplus b_1\dot{x}_1 \oplus b_2\dot{x}_2 \oplus b_3\dot{x}_2\dot{x}_1 \\ &\quad \oplus \cdots \oplus b_{2^c-1}\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \oplus b_{2^c}\dot{x}_{c+1} \\ &\quad \oplus b_{2^c+1}\dot{x}_{c+1}\dot{x}_1 \oplus b_{2^c+2}\dot{x}_{c+1}\dot{x}_2 \\ &\quad \oplus \cdots \oplus b_{2^{c+1}-2}\dot{x}_{c+1}\dot{x}_c\cdots\dot{x}_2 \\ &\quad \oplus b_{2^{c+1}-1}\dot{x}_{c+1}\dot{x}_c\cdots\dot{x}_1 \\ &= (b_0 \oplus b_{2^c}\dot{x}_{c+1}) \oplus (b_1 \oplus b_{2^c+1}\dot{x}_{c+1})\dot{x}_1 \\ &\quad \oplus (b_2 \oplus b_{2^c+2}\dot{x}_{c+1})\dot{x}_2 \\ &\quad \oplus \cdots \oplus (b_{2^c-1} \oplus b_{2^{c+1}-1}\dot{x}_{c+1})\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \end{aligned} \tag{3}$$

Defining:

$$B_j = b_j \oplus b_{2^c+j}\dot{x}_{c+1} = \begin{cases} 0 & \text{when } b_j = 0,\ b_{2^c+j} = 0 \\ 1 & \text{when } b_j = 1,\ b_{2^c+j} = 0 \\ \dot{x}_{c+1} & \text{when } b_j = 0,\ b_{2^c+j} = 1 \\ \bar{\dot{x}}_{c+1} & \text{when } b_j = 1,\ b_{2^c+j} = 1 \end{cases}$$

$$j = 0, 1, 2, \ldots, 2^c - 1$$

Then eqn. 3 can be written as follows:

$$\begin{aligned} f(x_{c+1}, x_c, \ldots, x_1) &= B_0 \oplus B_1\dot{x}_1 \oplus B_2\dot{x}_2 \\ &\quad \oplus \cdots \oplus B_{2^c-1}\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \end{aligned} \tag{4}$$

It can be seen that eqn. 4 is identical to eqn. 1 when B_j is equated to g_j. Hence the theorem is proved.

Although it is possible, in theory, to design large RM–ULMs to suit large expressions, it would be more practical to connect small RM–ULMs in the form of tree networks to realise functions having a large number of variables.

Theorem 3: Each data input of a RM–ULM(c) can be connected to an output of another module RM–ULM(d) to produce a network which consists of 2^c RM–ULM(d) and a single RM–ULM(c). This network is equivalent to a single RM–ULM(c + d).

Proof: Assume the data inputs of 2^c RM–ULM(d) are g_K, $k = 0, 1, \ldots, 2^{d+c} - 1$ (See Fig. 2). Their control inputs are $\dot{x}_{c+1}, \dot{x}_{c+2}, \ldots, \dot{x}_{c+d}$. According to the definition, their outputs are

$$\begin{aligned} f_0(d) &= g_0 \oplus g_1\dot{x}_{c+1} \oplus g_2\dot{x}_{c+2} \\ &\quad \oplus \cdots \oplus g_{2^d-1}\dot{x}_{c+d}\cdots\dot{x}_{c+1} \\ f_1(d) &= g_{2^d} \oplus g_{2^d+1}\dot{x}_{c+1} \oplus g_{2^d+2}\dot{x}_{c+2} \\ &\quad \oplus \cdots \oplus g_{2^{d+1}-1}\dot{x}_{c+d}\dot{x}_{c+d-1}\cdots\dot{x}_{c+1} \\ &\vdots \\ f_{2^c-1}(d) &= g_{2^{c+d-1}} \oplus g_{2^{c+d-1}+1}\dot{x}_{c+1} \oplus g_{2^{c+d-1}+2}\dot{x}_{c+2} \\ &\quad \oplus \cdots \oplus g_{2^{c+d}-1}\dot{x}_{c+d}\dot{x}_{c+d-1}\cdots\dot{x}_{c+1} \end{aligned}$$

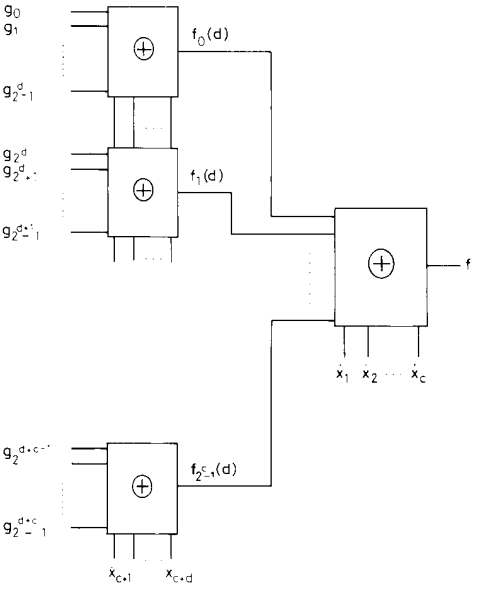

Fig. 2 *RM–ULM network*

As shown in Fig. 2, $f_i(d)$, $i = 0, 1, \ldots, 2^c - 1$ are the data inputs of the next level module RM–ULM(c). If $\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_c$ are control inputs of this module, the output f is given by

$$\begin{aligned} f &= f_0(d) \oplus f_1(d)\dot{x}_1 \oplus \cdots \oplus f_{2^c-1}(d)\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \\ &= g_0 \oplus g_1\dot{x}_{c+1} \oplus g_2\dot{x}_{c+2} \\ &\quad \oplus \cdots \oplus g_{2^d-1}\dot{x}_{c+d}\dot{x}_{c+d-1}\cdots\dot{x}_{c+1} \\ &\quad \oplus g_{2^d}\dot{x}_1 \oplus g_{2^d+1}\dot{x}_{c+1}\dot{x}_1 \oplus g_{2^d+2}\dot{x}_{c+2}\dot{x}_1 \\ &\quad \oplus \cdots \oplus g_{2^{d+1}-1}\dot{x}_{c+d}\dot{x}_{c+d-1}\cdots\dot{x}_{c+1}\dot{x}_1 \\ &\quad \oplus \cdots \oplus g_{2^{c+d}-1}\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \\ &\quad \oplus g_{2^{c+d-1}+1}\dot{x}_{c+1}\dot{x}_c\cdots\dot{x}_1 \\ &\quad \oplus \cdots \oplus g_{2^{c+d}-1}\dot{x}_{c+d}\dot{x}_{c+d-1}\cdots\dot{x}_{c+1}\dot{x}_c\dot{x}_{c-1}\cdots\dot{x}_1 \end{aligned} \tag{5}$$

According to the definition, the output of a RM–ULM(c + d) is

$$\begin{aligned} f(c + d) &= b_0 \oplus b_1\dot{x}_1 \oplus b_2\dot{x}_2 \\ &\quad \oplus \cdots \oplus b_{2^{c+d}-1}\dot{x}_{c+d}\dot{x}_{c+d-1}\cdots\dot{x}_{c+1}\cdots\dot{x}_1 \end{aligned} \tag{6}$$

When eqns. 5 and 6 are compared, they are seen to be identical. Hence the theorem is proved.

3 Optimisation of networks

Theorem 3 shows that RM expansions with a large number of variables can be implemented using small RM–ULMs in the form of trees similar to ULM trees implementing large Boolean functions. As with Boolean expressions, the number of modules required for RM–ULM trees depends on the selection of control variables used at different levels.

A programmed algorithm is described next to optimise RM–ULM networks. The algorithm works on the principle of level-by-level minimisation, selecting the control variables that result in the minimum number of continuing branches. This occurs when two identical branches are identified, and when a branch terminates with a single variable or a logical constant. A choice of control variables that result in at least one redundant branch at an early level (closer to the output) should be utilised, knowing that the chance of a better finding at a later stage is statistically very small indeed [2]. This is particularly true for large networks. Ultimately, a cascade connection with a single module in each level and only one branch continuing is given a preference, keeping in mind that it is also desirable to reduce the number of levels (see example 2).

3.1 Algorithm

Step 1: Express the piterms of the given function in decimal form (assume zero polarity). Get the number of variables n. Set the level $l = 1$.

Step 2: Convert the piterms $\{p_i\}$ according to the polarity specified by the user.

Step 3: Get the number of control variables per module c and check whether the number of variables prior to level l obeys the inequality $n - c(l-1) \leqslant c + 1$. If so the tree can finish with any choice of remaining variables. If not then continue.

Step 4: Check if there is any c-tuple of variables $\{x_{i_r}: r = 1, \ldots, c\}$ for which the number of simultaneous occurrences in the piterm table

$$S_c(\hat{x}_{i_1}, \ldots, \hat{x}_{i_c}) = 0 \tag{4a}$$

or $\exists p_i$ corresponding to $S_c(\hat{x}_{i_1}, \ldots, \hat{x}_{i_c}) = 1$ such that

$$p_i(k) = 0 \quad \forall k \notin \{i_1, \ldots, i_c\} \tag{4b}$$

If either eqn. 4*a* or eqn. 4*b* is true, increment the saved branch counter $y_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c})$. Keep a record of the input $I_c(\hat{x}_{i_1}, \ldots, \hat{x}_{i_c}) = 0$ or 1, respectively.

This step checks whether a module input is either 0 (eqn. 4*a*) or 1 (eqn. 4*b*). For single control modules, eqn. 4*a* is equivalent to testing whether for any control variable $\dot{x}_i$, $\dot{x}_i = 1$ throughout the piterm table ($S_1(\underline{\dot{x}}_i) = 0$) or whether $\dot{x}_i = 0$ ($S_1(\dot{x}_i) = 0$). Eqn. 4*b* checks whether corresponding to a particular module input there is only a single piterm and that for this piterm $\dot{x}_i = 0$ for all variables i which have not previously been selected (i.e. not previously used as a control variable at an earlier level).

Step 5: Check if there is any c-tuple of variables which satisfies

$\exists p_i$ corresponding to $S_c(\hat{x}_{i_1}, \ldots, \hat{x}_{i_c}) = 1$ such that $\exists k, k \notin \{i_1, \ldots, i_c\}$, $p_i(k) = 1$ and $p_i(m) = 0$ $\forall m \notin \{i_1, \ldots, i_c, k\}$, or (5*a*)

$\exists p_i, p_j$ corresponding to $S_c(\hat{x}_{i_1}, \ldots, \hat{x}_{i_c}) = 2$ such that $p_i(k) = 0$ $\forall k \notin \{i_1, \ldots, i_c\}$ and $\exists k$, $k \notin \{i_1, \ldots, i_c\}$ such that $p_j(k) = 1$ and $p_j(m) = 0$ $\forall m \notin \{i_1, \ldots, i_c, k\}$ (5*b*)

If eqn. 5*a* or eqn. 5*b* is true, increment the saved branch counter $y_c(\dot{x}_{i_1}, \ldots, \dot{x}_{i_c})$ and keep a record of the input $I_c(\hat{x}_{i_1}, \ldots, \hat{x}_{i_c}) = \dot{x}_k$ (eqn. 5*a*) or $= 1 \oplus \dot{x}_k$ (eqn. 5*b*).

This step checks whether a module input is either $\dot{x}_k$ (eqn. 5*a*) or $1 \oplus \dot{x}_k$ (eqn. 5*b*) where $\dot{x}_k$ is an arbitrary variable not already selected. Eqn. 5*a* is satisfied if corresponding to a particular module input there is only a single piterm and that for this piterm there is an unselected variable j, say, such that $\dot{x}_j = 1$ but for all other unselected variables i, $\dot{x}_i = 0$. Eqn. 5*b* checks whether corresponding to a particular module input there are two piterms, one which satisfies the requirements of eqn. 4*b* the other satisfying eqn. 5*a*.

Step 6: According to the results of steps 4 and 5, calculate which c-tuple of variables (to be chosen as controls) $\dot{x}_{h_1}, \ldots, \dot{x}_{h_c}$ maximises the number of saved branches.

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

Step 7: According to the choice made in step 6 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 those inputs separately and derive their piterm 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 3 and repeat the procedure.

3.2 Example 1:

Implement the following function using RM–ULM(1)s.

$$f_0 = x_1 \oplus x_2 \oplus x_2 x_1 \oplus x_3 x_2 x_1 \oplus x_4 x_2 x_1 \oplus x_4 x_3 x_2$$
$$= \oplus \sum (1, 2, 3, 7, 11, 14)$$

Step 1: $n = 4$, $l = 1$. Note that in this paper we refer to the first piterm by p_1, the second piterm by p_2, and so on. So that here $p_1 = 1$ and $p_6 = 14$, etc.

Step 2: Choose polarity zero.

Step 3: $c = 1$, $n - c(l-1) = 4 > 2$. The piterm table is given in Table 1.

Table 1: Piterms for example 1

	x_4	x_3	x_2	x_1
p_1	0	0	0	1
p_2	0	0	1	0
p_3	0	0	1	1
p_4	0	1	1	1
p_5	1	0	1	1
p_6	1	1	1	0

Steps 4 and 5 are carried out for possible controls $\dot{x}_i = x_i$, $i = 1, \ldots, 4$, and for each control, the number of inputs which have either variables coming in or fixed inputs 0 or 1, is determined.

Control x_1

Step 4: $S_1(\hat{x}_1) \neq 0, 1$, so eqns. 4*a* and 4*b* fail.

Step 5: $S_1(x_1) = 4$, $S_1(\underline{x}_1) = 2$; therefore eqns. 5*a* and 5*b* fail. No branches can be saved.

Control x_2

Step 4: $S_1(x_2) = 5$, so eqn. 4*a* fails. As $S_1(\underline{x}_2) = 1$ but $p_1(1) = 1$, eqn. 4*b* fails.

Step 5: $S_1(\underline{x}_2) = 1$ and $p_1(1) = 1$ and $p_1(3) = 0$, $p_1(4) = 0$. Consequently eqn. 5*a* is satisfied and so $y_1(x_2) = 1$, $I(\underline{x}_2) = x_1$.

It is found that no other choice of control variable saves more than one branch, thus x_2 is chosen as the control to the first module (step 6).

Step 7: Obtain the reduced subfunction for the input (Table 2). Increment the level. Go to step 3. $n - c(l-1) = 3 > 2$, so continue.

Table 2: Reduced piterm table for input x_2

	x_4	x_3	x_1
p_2	0	0	0
p_3	0	0	1
p_4	0	1	1
p_5	1	0	1
p_6	1	1	0

Step 4: Consider Table 2 with control x_1. $S_1(x_1) = 3$, $S_1(\underline{x}_1) = 2$, so eqns. 4*a* and 4*b* fail.

Step 5: As $S_1(\hat{x}_1) \neq 1$, eqn. 5*a* fails. $S_1(\underline{x}_1) = 2$ (p_2, p_6), but $p_6(4) = 1$ and $p_6(3) = 1$, therefore eqn. 5*b* fails.

It is found that no other choice of control variables saves any branches, thus x_1 is chosen as the control (step 6).

Step 7: Increment the level. Go to step 3, $n - c(l-1) = 2$. The inequality is satisfied so the tree can finish with any choice of remaining variables. The complete tree realisaton is shown in Fig. 3.

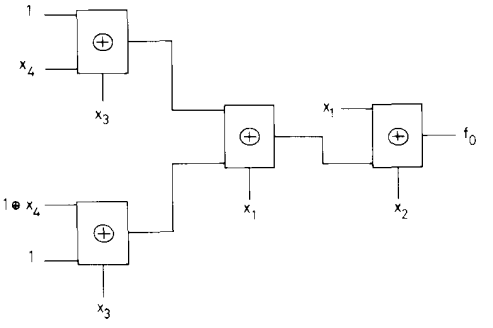

Fig. 3 *Optimised tree network for example 1*

3.3 Example 2

The nine variable function

$$f_0 = \oplus\sum(4, 6, 9, 10, 22, 32, 36, 38, 130, 278, 342, 406)$$

in polarity zero can be realised as a cascade using the optimisation algorithm (Fig. 4).

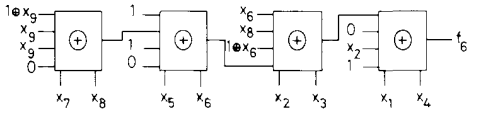

Fig. 4 *Cascade realisation of example 2*

4 Conclusions

The use of RM–ULMs for the implementation of fixed polarity GRM functions has been explained. It has been shown that small modules can be connected in a multi-level form to realise large functions. An algorithm for the minimisation of RM–ULM tree networks has been presented. The algorithm looks for possible cascade networks where only a single branch is continued into a new level. Where this cascade connection is not possible, a tree structure is accepted, but the level-by-level minimisation procedure selects the control variables that result in a minimum number of continuing branches. This ultimately minimises the number of modules required to implement a given function.

The algorithm is programmed using Fortran-77 and run on a DEC microVax II. Computation time for the examples given in this paper are all less than three seconds of CPU time. A large example of 20 variables and 200 piterms took 1.5 minutes of CPU time. The program was tested using RM–ULM(1) and RM–ULM(2) devices, but this can be generalised to any size. In fact, the algorithm allows mixed modules to be used should this be desirable. The program assumes fully specified fixed polarity GRM functions. It can handle any polarity and any number of variables.

Having run a large number of examples in different polarities, it was observed that for a given function, polarities having less piterms tend to give more economical networks in most, but not all, cases. Optimisation based on polarity is not addressed in this paper but is an area worth further investigation.

5 References

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

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

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

4 ALMAINI, A.E.A., MILLER, J.F., and XU, L.: 'Automated synthesis of digital multiplexer networks', *IEE Proc. E., Comput. Digit. Tech.*, 1992, **139**, (4), pp. 329–334

5 REDDY, S.M.: 'Easy testable realisations for logic functions', *IEEE Trans.*, 1972, **C-21**, pp. 1183–1188

6 WU, X., CHEN, X., and HURST, S.L.: 'Mapping of Reed–Muller coefficients and the minimisation of exclusive-OR switching functions', *IEE Proc. E., Comput. Digit. Tech.*, 1982, **129**, (1), pp. 15–20

7 HARKING, B.: 'Efficient algorithm for canonical Reed–Muller expansions of Boolean functions', *IEE Proc. E., Comput. Digit. Tech.*, 1990, **137**, (5), pp. 366–370

8 ALMAINI, A.E.A., THOMSON, P., and HANSON, D.: 'Tabular techniques for Reed–Muller logic', *Int. J. Electron.*, 1991, **70**, (1), pp. 23–34

9 GREEN, D.: 'Modern logic design' (Addison Wesley, 1986)