Symbolic method for simplifying AND-EXOR representations of Boolean functions using a binary-decision technique and a genetic algorithm

P. Thomson
J.F. Miller

Indexing terms: ESOP optimisation, Reed–Muller logic, Genetic algorithms, Binary-decision diagrams

Abstract: An algorithm called XORGA is presented which minimises Boolean multi-output logic functions as multilevel AND-EXOR networks of two-input logic gates. It carries out symbolic simplification, and works from the bottom of a binary variable decision tree to the top, with variable choice determined using a genetic algorithm. Since the algorithm is multi-level in nature, it delivers more compact circuits than two-level ESOP minimisation algorithms, such as EXMIN2. It also finds more economical representations than the fixed polarity Reed–Muller method.

1 Introduction

The binary decision diagram (BDD) method is now widely used for the optimisation of combinational Boolean logic [1–5]. The number of nodes required in the BDD, and hence the compactness of the function representation, is strongly dependent on the order chosen for the decision variables. Many algorithms exist which attempt to find a good variable ordering [6–11]. The set of all possible variable orderings is exponentially dependent on the number of variables. In this paper, we recommend the use of a genetic algorithm (GA) to determine a good variable ordering. GAs have been widely applied to problems involving very large search spaces [12–15].

Reed–Muller implementations of Boolean logic functions [16–24] are on average more compact than Boolean [25, 26], in addition, they possess the well known advantage of improved testability [27]. Traditionally, Reed–Muller designs have been seen as more costly to implement, however, modern FPGA devices have now removed this cost barrier.

The BDD technique has not yet been widely applied to the problem of optimising combinational logic functions in the Reed–Muller domain and, indeed, attempts have largely been restricted to the fixed-polarity Reed–Muller expansions [28, 29], although Sasao [26] has used ternary decision diagrams to optimise the pseudo-Kronecker (PSDKRO) form, and Yasuoka [30] devised a method using shared BDDs. The use of BDDs to represent *fixed polarity* representations is questionable, as there is evidence to suggest that there are more computationally efficient techniques available [31]. However, BDD techniques are more useful for the more general Reed–Muller representations, such as the psuedo-Kronecker form (PSDKRO) [26] and the exclusive-OR sums-of-products (ESOPs) [26, 32]. ESOPs represent the most general form of expansion, in that all other possible expansion sets are subsets, hence optimisation algorithms which can deal with this form are likely to find the most economical representations. The technique presented in this paper was originally inspired by the BDD method. We wished to avoid the expansion associated with the top-down BDD approach, and so we inverted the BDD tree and thought about the problem as being one of contraction. We therefore had to develop symbolic simplification rules, which could effectively prune the decision tree automatically (these rules are discussed in the next Section). To easily identify common symbolic sub-functions, we replaced them with new symbols (i.e. factors). Hence we produced an algorithm which naturally produced a multi-level logic representation. The objective of the technique was to produce an economical symbolic representation of the input function, and not a graphical representation of it. It is well known that there are some functions which cannot be economically represented as BDDs [33]. These functions can still be simplified using our method. We have not yet extended the method to incompletely specified functions, but work is progressing on this topic.

2 Symbolic method for simplifying AND-EXOR sums-of products and use of a genetic algorithm to resolve the variable ordering problem

Implicit in our algorithm XORGA is the concept of the ordered binary decision diagram (OBDD), in which each level of a BDD tree employs a single input variable, which can only occur in a single level in the whole tree. We use a genetic algorithm to determine a good ordering for these variables. However, the BDD is only used as a framework for the creation of symbolic expressions, which represent the input function. In XORGA, symbolic sub-representations of the input function are built-up from the bottom of the variable decision tree to the top. We do not attempt to produce

IEE Proceedings online no. 19960196

Paper first received 28th March 1995 and in revised form 25th October 1995

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

decision trees with a minimum number of nodes and, unlike the standard BDD technique, we do not carry out tree-pruning operations.

A top-down simplification of an OBDD can sometimes be inefficient, in that a large number of operations are required to identify and remove equivalent subfunctions. A downwards search, followed by a return to the original BDD level, must be conducted to identify equivalence. In our algorithm, simplification proceeds in one direction only, from the bottom of the variable ordering tree to the top, equivalent subfunctions are identified using the symbolic simplification rules described later.

The number of possible variable orderings is $n!$, where n is the number of input variables. We employ a genetic algorithm to determine a good ordering of the input variables. The genetic algorithm is a non-deterministic method of optimisation first developed by Holland [13]. There are now numerous standard texts describing the basic technique [11–13]. In our GA, a chromosome is defined as a 'pick' or ordinal list. A permutation $\{p_i\}$ of n variables chosen from $N = \{1, ..., n\}$ is obtained from an ordinal chromosome $\{c_i\}$ where $i = 1$ to n, in the following way: p_i is the c_ith item chosen from $N - \{p_k\}$, where $k = 1$ to $i - 1$. The subtraction denotes the set theoretic difference operation. The ordinal chromosome representation automatically guarantees that crossover will produce valid permutations. We have found this representation to work quite well for other permutational search spaces [34]. We also employed a technique known as elitism, in which the most fit chromosome of a generation is automatically promoted to the next generation. The significant difference between our approach to the variable ordering problem and other BDD ordering strategies is that the GA uses the actual implementation cost as the discriminator between different possible solutions, rather than some estimate of what it might be. A chromosome with fitness f_i and gate count g_i, in a population with worst gate count g_{worst}, was defined by $f_i = g_{worst} - g_i$. We then selected chromosomes with a probability proportional to their fitness ('roulette wheel'). The gate count associated with a chromosome is merely the sum of the two-input AND and EXOR gates that would be required to implement the resultant circuit. The use of two-input logic gates in no way precludes the use of multiple-input gates at the time of actual circuit implementation, but is merely used here as an abstract measure of implementation cost. We stress this as we feel that the literature lacks a standard performance measure, thus making comparison between various algorithms extremely difficult.

The AND-EXOR multilevel simplification algorithm XORGA is presented in Fig. 1.

It should be noted that the function minterms are represented as binary strings. The algorithm then operates by considering matched pairs of consecutive even and odd minterm strings, upon which it is able to perform simplification according to the set of rules presented below. When no pairing is possible, the algorithm operates upon an isolated even or odd term. After each iteration is complete, the addresses are updated by losing their rightmost bit. The operator >> 1 is used to show this.

The set of result strings calculated at each stage may be used to form a partially simplified representation of the input function, the fully simplified representation being obtained on the final iteration.

A binary-string representation of minterm addresses is used, so that the algorithm will be able to handle very large numbers of input variables, provided the number of minterms is not too large. This circumvents the difficulty which a limited integer size can cause.

The simplification rules used here are based upon the pairing of minterm strings. Consider two result strings, R and S, which were combined under ordering variable

```
Step 1: Store the input function truth table as minterm strings.
Step 2: Minterm strings re-arranged according to the selected variable ordering.
Step 3: for (each variable in the selected order)
            for (each minterm string)
            {
               if (minterm string is of even) then
                  if (next highest minterm string is consecutive) then
                  {
                     employ Reed–Muller simplification rules on result strings
                     corresponding to minterm string pairs to find new result string;
                     store new result string with corresponding (minterm string) >> 1;
                  }
                  else
                  {
                     employ Reed–Muller simplification rules on result string
                     corresponding to isolated even minterm string to calculate
                     new result string;
                     store new result string with corresponding (minterm string) >> 1;
                  }
               else
               {
                  employ Reed–Muller simplification rules on result string
                  corresponding to isolated odd minterm string to calculate
                  new result string;
                  store new result string with corresponding (minterm string) >> 1;
               }
               update minterm string as (minterm string) >> 1;
            }
Step 4: Calculate number of two-input gates required to implement result.
```

Fig. 1 *The AND-EXOR simplification algorithm XORGA*

x_{n-1}, and are now to be combined under ordering variable x_n, to produce result string, T, then in general if

$$R = A.\bar{x}_{n-1} \oplus B.x_{n-1} \oplus E \quad (1)$$

$$S = C.\bar{x}_{n-1} \oplus D.x_{n-1} \oplus F \quad (2)$$

where A, B, C, D, E and F are functions of previous ordering variables, then T will be given by

$$T = R.\bar{x}_n \oplus S.x_n \quad (3)$$

and after the use of the simplification rules, if applicable, T will be given by

$$T = A'.\bar{x}_n \oplus B'.x_n \oplus E' \quad (4)$$

Depending on the commonality of string coefficients A, B and C with D, E and F, the actual expression which represents T assumes a simpler form according to the rules shown in Table 1.

Table 1

Rule 1: $A = C$ only	Rule 2: $B = D$ only
$A' = B.x_{n-1} \oplus E$	$A' = A.\bar{x}_{n-1} \oplus E$
$B' = D.x_{n-1} \oplus F$	$B' = C.\bar{x}_{n-1} \oplus F$
$E' = A.x_{n-1}$	$E' = B.x_{n-1}$
Rule 3: $E = F$ only	Rule 4: $A = C$ and $B = D$
$A' = A.\bar{x}_{n-1} \oplus B.x_{n-1}$	$A' = E$
$B' = C.\bar{x}_{n-1} \oplus D.x_{n-1}$	$B' = F$
$E' = E$	$E' = A.\bar{x}_{n-1} \oplus B.x_{n-1}$
Rule 5: $A = C$ and $E = F$	Rule 6: $B = D$ and $E = F$
$A' = B.x_{n-1}$	$A' = A.\bar{x}_{n-1}$
$B' = D.x_{n-1}$	$B' = C.\bar{x}_{n-1}$
$E' = A.\bar{x}_{n-1} \oplus E$	$E' = B.x_{n-1} \oplus E$
Rule 7: $A = C$, $B = D$ and $E = F$	Rule 8: $A = D$ only
$A' = 0$	$A' = B.x_{n-1} \oplus D \oplus E$
$B' = 0$	$B' = C.\bar{x}_{n-1} \oplus F$
$E' = A.\bar{x}_{n-1} \oplus B.x_{n-1} \oplus E$	$E' = D.x_{n-1}$
Rule 9: $B = C$ only	Rule 10: $A = D$ and $E = F$
$A' = A.\bar{x}_{n-1} \oplus E$	$A' = B.x_{n-1} \oplus D$
$B' = D.x_{n-1} \oplus B \oplus F$	$B' = C.\bar{x}_{n-1}$
$E' = B.x_{n-1}$	$E' = D.x_{n-1}$
Rule 11: $B = C$ and $E = F$	Rule 12: $A = D$ and $B = C$
$A' = A.\bar{x}_{n-1}$	$A' = D \oplus E$
$B' = D.x_{n-1} \oplus B$	$B' = B \oplus F$
$E' = B.x_{n-1} \oplus E$	$E' = (D \oplus B).x_{n-1}$
Rule 13: $A = D$, $B = C$ and $E = F$	Rule 14: No simplification possible
$A' = A$	$A' = A.\bar{x}_{n-1} \oplus B.x_{n-1} \oplus E$
$B' = B$	$B' = C.\bar{x}_{n-1} \oplus D.x_{n-1} \oplus F$
$E' = (A \oplus B).x_{n-1} \oplus E$	$E' = 0$

Throughout these expressions, further simplification is performed whenever possible. For example, in Rule 14, if $B = E = 1$, then clearly further simplification is possible. Additionally, whenever a new term appears which contains at least one exclusive-OR, then this is replaced by a symbolic factor, and the string corresponding to the factor is added to a list of factors. If that same factor then appears at a later stage in the simplification, then this is entered symbolically into the current expression. This means that if a particular factor is already present, and has been simplified, then it need not be physically generated again in the final circuit implementation.

Example 1: Consider the following single-output function (the objective function is the seventh output of benchmark 'bw', designated as 'bw7') which has minterms

$$f = \sum(6, 11, 12, 13, 14, 17, 18, 19, 24, 25, 26, 27)$$

The algorithm would produce, after each iteration, the following A, B and E strings for the variable ordering 4, 2, 3, 1, 0:

Step 1: Minterm strings are (in decimal form) 6, 11, 12, 13, 14, 17, 18, 19, 24, 25, 26 and 27.

Step 2: These are mapped according to variable ordering as 5, 6, 9, 10, 13, 14, 17, 21, 22, 25, 28 and 29.

Step 3: see Table 2.

Table 2: Step 3

0th iteration: applying simplification rule 14 only (no minterm pairs)

Address	A	B	C
5	0	0	1
6	0	0	1
9	0	0	1
10	0	0	1
13	0	0	1
14	0	0	1
17	0	0	1
21	0	0	1
22	0	0	1
25	0	0	1
28	0	0	1
29	0	0	1

1st iteration: applying simplification rules 12 and 14

Address	A	B	C
2	0	1	0
3	1	0	0
4	0	1	0
5	1	0	0
6	0	1	0
7	1	0	0
8	0	1	0
10	0	1	0
11	1	0	0
12	0	1	0
14	0	0	1

2nd iteration: applying simplification rule 14, 7, 9 and 6

Address	A	B	E
1	0	1	x_4
2	0	1	x_4
3	0	1	x_4
4	x_4	0	0
6	x_4	0	0
7	1	0	0

3rd iteration: applying simplification rules 1 and 5 where $P_0 = x_2 \oplus x_4$

Address	A	B	E
0	0	P_0	0
1	0	0	P_0
2	$x_4.\bar{x}_2$	P_0	0
	$x_4.\bar{x}_2$	$\bar{x}_2$	0

4th iteration: apply simplification rules 1

Address	A	B	E
0	$P_0.x_3$	P_0	0
0	$P_0.x_3$	$\bar{x}_2.x_3$	$x_4.\bar{x}_2.\bar{x}_3$

5th iteration: procedure finished where $P_1 = \bar{x}_2.x_3.x_1 \oplus x_4\bar{x}_2.\bar{x}_3$

Address	A	B	E
0	$P_0.x_1$	P_1	$P_0.x_3.\bar{x}_1$

So, the simplified function is:

$$f = P_0.x_1.\overline{x}_0 \oplus P_1.x_0 \oplus P_0.x_3.\overline{x}_1 \quad (5)$$

3 Results

XORGA was written in C and run on an HP-720 workstation. The GA parameters were as follows: population size 10, number of generations = 20, breeding rate = 60.0% and mutation rate = 10.0%, where the breeding rate is the percentage of the population chromosomes which undergo crossover and replacement by their offspring, and the mutation rate is the percentage of population genes which are chosen to be randomly altered. These values were settled upon by experience. The mutation rate is somewhat high for a GA application, but we found that for some benchmark functions, the GA has little effect, whereas for others, the outcome is extremely sensitive to variable ordering. On the benchmarks tested, the program took of the order of minutes to execute. For example, the worst case runtime '*t*3' was 530 seconds. In Table 3 we present the results of applying the algorithm to various multi-output MCNC and ESPRESSO benchmark logic functions, and for comparison we present the two-level AND-EXOR two-input gate counts produced by EXMIN2 [35].

Table 3: Gate counts (2-input) for XORGA (multi-level) and EXMIN2 (two-level)

Benchmark	Inputs/ outputs		EXMIN2 (2-level)		XORGA (multi-level)	
			XOR Gates	AND Gates	XOR Gates	AND Gates
5xp1	7	10	61	125	49	103
clip	9	5	113	404	110	208
life	9	1	54	361	23	51
rd53	5	3	19	41	14	21
adr4	8	5	40	122	20	23
mlp4	8	8	84	311	119	268
log8mod	8	5	164	526	48	95
sao2	10	4	62	246	67	161
t3	12	8	43	166	22	113
rd73	7	3	56	165	38	58
rd84	8	4	67	263	55	90
z4	7	4	34	111	22	25
sym10	10	1	84	667	44	71
t481	16	1	13	40	12	57
sqr6	6	12	201	546	44	97

Table 4 demonstrates that AND-EXOR mutilevel solutions are generally much more compact than fixed polarity Reed–Muller expansions, even if these are exhaustively searched [19, 21–23]. In Table 5 we list the performance of XORGA on a further set of benchmarks.

It should be noted that the results presented in the Tables for XORGA are all verified as being logically equivalent to the original benchmark functions.

One drawback of the method is that it requires an initial minterm set for each output to be considered. However, it is the use of a full minterm description of the function, as opposed to a set of disjoint implicants, as employed by some other methods, coupled with the use of the genetic algorithm, that we feel, increases the effectiveness of the algorithm. Methods for dealing with the PLA implicant representation of the functions, the form in which benchmarks are usually presented, which do not require a conversion to minterms, are currently being investigated.

Table 4: Comparison of XORGA with fixed polarity (FPEX)

Benchmark	Variables/ minterms		FPEX (fixed polarity exhaustive research)		XORGA	
			XOR Gates	AND Gates	XOR Gates	AND Gates
5xp1	7	52	11	34	6	12
9sym	9	420	172	464	35	55
bw7	5	12	7	13	4	9
con12	7	88	7	13	3	4
f51m4	8	128	6	9	6	6
life	9	140	99	496	23	51
max46	9	62	205	820	44	155
newill	8	142	12	57	7	19
newtag	8	234	5	22	7	9
rd532	5	16	4	0	4	0
rd842	8	128	7	0	7	0
sao22	10	10	51	322	15	44
sao23	10	476	46	286	18	38
sym10	10	837	265	1034	44	71
xor5	5	16	4	0	4	0
z42	7	52	8	13	9	10

Table 5: Further XORGA results

Benchmarks	Inputs/outputs		XOR Gates	AND Gates
Z5xp1	7	10	46	102
addm4	9	8	156	335
con1	7	2	9	16
dc1	4	7	16	35
dc2	8	7	34	100
f51m	8	8	58	105
m1	6	12	28	67
m2	8	16	90	209
m4	8	16	175	377
max46	9	1	44	155
max512	9	6	145	326
misex1	8	7	22	60
newcpla1	9	16	31	110
newcpla2	7	10	23	74
newcwp	4	5	9	10
newill	8	1	7	19
newtag	8	1	7	9
newtpla1	10	2	2	18
newtpla2	10	4	10	41
p82	5	14	29	105
radd	8	5	33	57
rd73x	7	3	38	58
root	8	5	53	120
sex	9	14	20	67
sqn	7	3	36	76
sqrt8	8	4	28	52
squar5	5	8	18	44

4 Conclusions

A BDD inspired symbolic simplification method for AND-EXOR multilevel implementations of Boolean functions has been presented. A genetic algorithm has been used to resolve the variable ordering problem. The algorithm produces solutions suitable for implementation on devices such as field-programmable gate arrays (FPGAs). In terms of the number of two-input AND and XOR gates used to implement these solutions, the method produces more economical circuits than two-level minimisers such as EXMIN2.

Two areas in which the use of a multi-level, factor-based implementation method, as described here, may cause concern are the possibility of the introduction of additional race hazards due to multi-path delays within the final circuit, the possibility that a factor output, which has to be realised physically, may be driving too many subsequent inputs, causing a technology-dependent fan-out violation.

It is our intention, in a further refinement of the genetic algorithm, to decrease the fitness associated with chromosomes which cause multi-path delay and fan-out problems, and thus favour solutions which meet the user-defined constraints.

Clearly the restriction of the algorithm to the minterm form of specification is a significant drawback. However, recent work suggests that the algorithm may be readily extended to deal with implicants. Our findings in this regard will be reported in due course.

Finally, we found that looking at the ESOP minimisation problem from the point of view of evolutionary optimisation techniques clarified our thinking about the problem in that, because we felt the GA would find a good variable ordering, we could subsequently consider more novel ways to minimise the input function as an ESOP, and hence provide the GA with a fitness function. From this starting point we developed the symbolic 'bottom-up' simplification algorithm.

5 References

1 ABORHEY, S.: 'Binary decision graph reduction', *IEE Proc. E*, 1989, **136**, (4), pp. 277–283

2 AKERS, S.B.: 'Binary decision diagrams', *IEEE Trans. Comput.*, 1978, **C-27**, (6), pp. 509–516

3 BRACE, K.S., RUDELL, R.L., and BRYANT, R.B.: 'Efficient implementation of a BDD package', 27th ACM/IEEE *Design automation* conference, 1990, pp. 40–45

4 BRYANT, R.B.: 'Graph-based algorithms for boolean function manipulation', *IEEE Trans. Comput.*, 1986, **C-35**, (8), pp. 677–691

5 ISHIURA, N.: 'Synthesis of multilevel logic circuits from binary decision diagrams', *IEICE Trans. Inf. Syst.*, 1993, **E-76D**, (9), pp. 1085–1092

6 BESSON, T., BOUZOUZOU, H., FLORICICA, I., SAUCIER, G., and ROANE, R.: 'Input order for ROBDDS based on kernel analysis', EuroASIC, EDAC, 1993,

7 FRIEDMAN, S.J., and SUPOWIT, K.: 'Finding the optimal variable ordering for binary decision diagrams', *IEEE Trans. Comput.*, 1990, **C-39**, (5), pp. 710–713

8 FUJITA, M., and MATSUNAGA, Y.: 'Variable ordering of binary decision diagrams for multilevel logic minimization', *Fujitsu Sci. Tech. J.*, 1993, **29**, pp. 137–145

9 ISHIURA, N., SAWADA, H., and YAJIMA, S.: 'Minimization of binary decision diagrams based on exchanges of variables', IEEE Int. Conf. on *CAD*, 1991, pp. 472–475

10 MINATO, S.: 'Minimum-width method of variable ordering for binary decision diagrams', *IEICE Trans. Fundam.*, 1992, **E-75-A**, pp. 392–399

11 YEH, F.-M., and LIN, C.-S.: 'Building BDDs with ordering reshuffle strategy', *Electron. Lett.*, 1993, **29**, (17), pp. 1540–1541

12 DAVIS, L. (Ed.): 'The handbook of genetic algorithms' (Van Nostrand Rheinhold, New York, 1991)

13 GOLDBERG, D.E.: 'Genetic algorithms in search, optimization and machine learning' (Addison–Wesley, Reading, Mass, 1989)

14 HOLLAND, J.H.: 'Adaptation in natural and artificial systems' (University of Michigan Press, Ann-Arbor, Mich, 1975)

15 MILLER, J.F., LUCHIAN, H., BRADBEER, P.V.G., and BARCLAY, P.J.: 'Using a genetic algorithm for optimizing fixed polarity Reed–Muller expansions of boolean functions', *Int. J. Electron.*, 1994, **76**, (4), pp. 601–609

16 ALMAINI, A.E.A.: 'Electronic logic systems' (Prentice–Hall, Englewood Cliffs, NJ, 1994, 3rd edn.)

17 EVEN, S., KOHAVI, I., and PAZ, A.: 'On minimal modulo-2 sums of products for switching functions', *IEEE Trans.*, 1967, **EC-16**, pp. 671–674

18 GREEN, D.H.: 'Modern logic design' (Addison–Wesley, Reading, Mass, 1987)

19 HARKING, B.: 'Efficient algorithm for canonical Reed–Muller expansion of boolean functions', *Proc. IEE E, Comput. Digit. Tech.*, 1990, **137**, (4), pp. 366–370

20 HELLIWELL, M., and PERKOWSKI, M.A.: 'A fast algorithm to minimize multi-output mixed polarity generalized Reed–Muller forms', 25th ACM/IEEE *Design Automation* Conference, 1988, pp. 427–432

21 LUI, P.K., and MUZIO, J.C.: 'Boolean matrix transformations for the parity spectrum and minimisation of modulo-2 canonical expressions', *IEE Proc. E*, 1991, **138**, pp. 411–417

22 MILLER, J.F., and THOMSON, P.: 'Highly efficient exhaustive search algorithm for optimizing canonical Reed–Muller expansions of boolean functions', *Int. J. Electron.*, 1994, **76**, (1), pp. 37–56

23 SARABI, A., and PERKOWSKI, M.A.: 'Fast exact and quasi-minimal minimization of highly testable fixed-polarity AND/XOR canonical networks', 29th ACM/IEEE *Design automation* conference, 1992, pp. 30–35

24 SASAO, T.: 'Logic synthesis and optimization' (Kluwer Academic Publishers, Mass., 1993, Chapter 12)

25 SASAO, T., and BESSLICH, PH.W.: 'On the complexity of Mod-2 sum PLAs', *IEEE Trans. Comput.*, 1990, **C-39**, (2), pp. 263–266

26 SASAO, T.: 'Logic synthesis and optimization' (Kluwer Academic Publishers, Mass., 1993, Chapter 13)

27 REDDY, S.M.: 'Easily testable realizations for logic functions', *IEEE Trans. Comput.*, 1972, **C-21**, pp. 1183–1188

28 MCKENZIE, L., XU, L., and ALMAINI, A.E.A.: 'Graphical representation of generalised Reed–Muller expansions', Proc. of IFIP Workshop on *Applications of Reed–Muller expansion in circuit design*, Hamburg, Germany, September 1993

29 PURWAR, S.: 'An efficient method of computing generalised Reed–Muller expansions from binary decision diagram', *IEEE Trans. Comput.*, 1991, **C-40**, (11), pp. 1298–1301

30 YASUOKA, K.: 'A generation method for EXOR-sum-of-products expressions using shared binary decision diagrams', in 'Logic synthesis and optimization' (Kluwer Academic Publishers, Mass., 1993, Chapter 14)

31 VINNAKOTA, B., and BAPESWARA RAO, V.V.: 'Generation of all Reed–Muller expansions of a switching function', *IEEE Trans. Comput.*, 1994, **C-43**, (1), pp. 122–124

32 SAUL, J., ESCHERMANN, B., and FROESSL, J.: 'Two-level logic circuits using EXOR sum of products', *IEE Proc. E*, 1993, **140**, (6), pp. 348–356

33 BRYANT, R.B.: 'On the complexity of VLSI implementations and graph representations of Boolean functions with applications to integer multiplication', *IEEE Trans. Comput.*, 1991, **40**, (2), pp. 205–213

34 ALMAINI, A.E.A., MILLER, J.F., THOMSON, P., and BILLINA, S.: 'State assignment of finite state machines using a genetic algorithm', *IEE Proc. Comput. Digit. Tech.*, 1995, **142**, (4), pp. 279–286

35 SASAO, T.: 'EXMIN2: a simplification algorithm for exclusive-OR sum-of-products expressions for multiple-valued-input two-valued-output functions', *IEEE Trans. Comput. Aided Des. Integrated Circuits Syst.*, 1993, **12**, (5), pp. 621–632