Journal of Computer Science 2 (7): 595-599, 2006
ISSN 1549-3636
© 2006 Science Publications

Single Stuck-At Fault Diagnosing Circuit of Reed-Muller Canonical Exclusive-Or Sum of Product Boolean Expressions

[1]P.N. Neelakantan and [2]A. Ebenezer Jeyakumar
[1]Electrical Engineering, Govt. College of Technology, Coimbatore-641 013, India
[2]Govt. College of Engineering, Salem – 636 011, India

Abstract: A testable design with a universal test set for single stuck-at zero and stuck-at one faults of Reed-Muller canonical form of Exclusive-OR sum of product logic expressions is proposed. The test circuit detects almost all the single stuck-at faults and needs only simple modifications for variations in the circuit under test. The number of test vectors is also quite small compared with the classical method. The factor of un-identifiability is discussed and a new quantification parameter for the fault diagnosis has also been introduced. Results of Matlab simulations for a few logic functions are included.

Key words: Combinational circuits, exclusive-or sum of products, Reed-Muller canonical form, single stuck-at faults, testability realization, universal test set

INTRODUCTION

Any arbitrary binary logic function can be expressed as exclusive-or sum of product Reed-Muller canonical (ESOP RMC) form which results in minimal product terms as can be seen from Table 1[1]. The SOP is the conventional sum of product form while the other forms are variations of Reed-Muller canonical (RMC) expressions. The PPRM is the positive-polarity RMC form, which does not allow any complemented variable to occur in the expression. For example, x1 ⊕ x2x3 ⊕ x4x5x6 is a PPRM expression, while x1' ⊕ x2x3 is not. The FPRM allows negation of any variable, but throughout the expression the variable should appear only in the same form, either complemented or uncomplemented. Thus, x1' ⊕ x2x3 ⊕ x1'x4 is a FPRM expression since x1 is appearing as only complemented variable, whereas the expression x1 ⊕ x2x3 ⊕ x1'x4 is not FPRM as x1 is present in uncomplemented form in the first term and in complemented form in the third term. GRM is the abbreviation for Generalized Reed-Muller form. In this structure, a variable is free to appear as complemented or uncomplemented, but should should not result in same PPRM terms more than once. For instance, x1 ⊕ x2x3 ⊕ x2'x3' is not a GRM since the term x2'x3' results in x2x3 when converted into PPRM, which is already present as the second term. The Exclusive-Or Sum of Product (ESOP) form, on the other hand, does not impose any of the restrictions mentioned above and, in fact, is the most general form of RMC expressions. Such an expression is of the form $f = a_0 \oplus a_1x_1^* \oplus a_2x_2^* \ldots\ldots \oplus a_nx_n^* \oplus a_{n+1}x_1^*x_2^* \oplus \ldots\ldots \oplus a_{2n-1}x_1^*x_2^*\ldots x_n^*$, where x_n^* can be x_n or its negation and a_n is either 0 or 1. The main advantage of such a form, apart from minimal number of product terms, is that it enables a simple method of diagnosis[2-4]. They also provide a more efficient realization than conventional AND-OR functions in many applications such as linear circuits, arithmetic circuits and telecom networks[5]. Further, a more compact PLA implementation based on AND-EXOR form is achievable compared with the AND-OR circuits[6]. The basic disadvantage of slow speed and greater chip area of exclusive-or based implementations has become less prominent, with the abundant availability of FPGA's since the last decade[7]. A Reed-Muller canonical form of CMOS implementation can be easily tested for stuck-open faults with a universal test set[8]. Mixed polarity Reed-Muller expressions have also been useful in classification of Boolean functions[11]. In spite of the slow speed and larger chip area of RMC implementations compared to other others, some of the RMC forms require only a lesser area and also have been effectively used in the FPGA based modules of Xilinx, Actel[9].

Table 1: Number of product terms of some arithmetic functions in different forms

Function	SOP	PPRM	FPRM	GRM	ESOP
adr4	75	34	34	34	31
log8	123	253	193	105	96
nrm4	120	216	185	96	69
ndm8	76	56	56	31	31
rot8	57	225	118	51	35
sym9	84	210	173	126	51
wgt8	255	107	107	107	58

Corresponding Author: P.N. Neelakantan, Electrical Engineering, Govt. College of Technology, Coimbatore-641 013, India

Literature survey: A classical method of generating test patterns for very large and complex logic functions is Linear Feedback Shift Register (LFSR) based pseudo-exhaustive or pseudo-random type[1]. However, this does not work well with ESOP form as shown by Drechsler *et al.*[10].A PPRM network for detection of stuck-at faults with a universal test of size n+4, n being the number of data inputs, was proposed by Reddy[2]. Though quite good for self-testing, the method is economical only for PPRM form, which obviously has more number of product terms than the other forms in most cases. Multiple stuck-at fault detection for ESOP circuits was carried out by Pradhan[11]. However since the cardinality is $2n+6+\sum nCe$, e= 0 to j, the order of ESOP expression, the test set is not universal and also is too large to be practical for large input functions. Stuck-at and bridging faults with a universal test set for PPRM has also been reported[12]. Multiple fault detecting GRM realizations was propounded by Sasao[4]. It was shown that 2n+s+3 test vectors, where s is the number of product terms in the logic function are required for single stuck-at fault detections in GRM circuit while 2n+s vectors are required for detection of and/or bridging faults in GRM/ESOP circuits[13]. Here too, the test set is not universal as it depends on s, the number of product terms of the function. Kalay *et al.*[1] described an ESOP implementation with a universal test set of size n+6 for single faults. A robust and universal sequence has been proposed for stuck-open type of faults in GRM/ESOP cmos transistor implementations[14]. Zhongliang[15] demonstrated that the single stuck-at fault detection can be achieved with only n+5 test vectors. Apart from a small modification in his circuit, two methods, each with minor modifications in his scheme, are proposed in this paper and results of matlab simulations for a few specific functions comparing the detectability of the faults have been included. Further, the concept of indistiguishability index has also been introduced and compared for the illustrative functions.

Network structure: The network structure of the proposed scheme is similar to that proposed by Zhongliang[15] and is shown in Fig.1. It comprises literal-complementing xor block, an AND block, an xor function tree block, which implements the required logic function as also two additional outputs o1 and o2 obtained through a separate AND and an OR gate. The actual data inputs to the system are x_1, x_2, …. x_n. Additionally, the scheme requires four control inputs c_0 to c_3. The literal-complementing block produces the complements of the literals used in the function. Only those literals appearing in complemented form require an xor gate in this block. The literals of each product term are combined through an AND gate and hence the number of AND gates required is the same as the number of product terms in the logic function. Further, each of the AND gates of this block may have an additional input from one of the control lines depending on the number of gates used in the xor tree block producing the final function f. For a function requiring seven xor gates as shown in Fig. 2, the eight AND gates connected to the eight input lines of the xor tree will receive additional control lines respectively from c_3, c_1, c_1, c_2, c_1, c_2, c_2 and c_3 respectively. All the product terms are the passed on to the function xor tree block, which generates the required logic function f. Finally, all the data and control inputs are applied to a separate AND gate and an OR gate, producing auxiliary outputs o1 and o2, to aid in the detection of faults which cannot be differentiated by the main function output f alone.

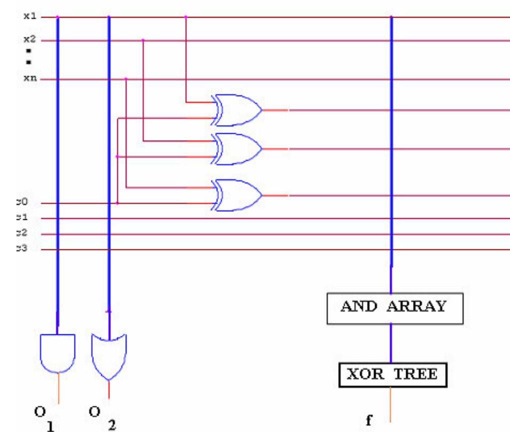

Fig. 1: Generalized network structure

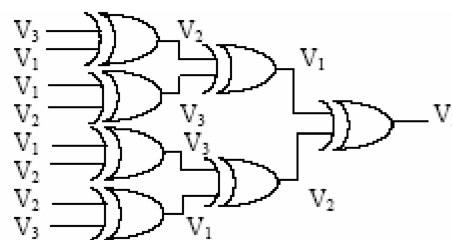

Fig. 2: An ex-or tree

Test vectors: Zhongliang[15] proposed a test matrix for the detection of single stuck-at faults. Each of its rows is an n+4 long vector, n being the number of data inputs. The first four columns of the matrix represent

the control inputs c_0 to c_3 while the remaining n columns that of the data inputs x1 to xn. The first test

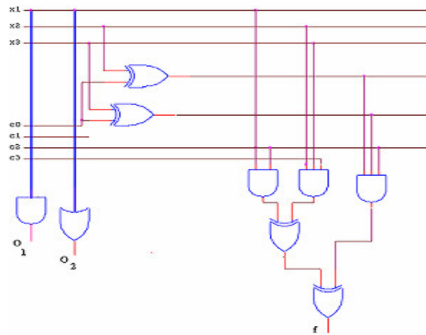

Fig. 3: Circuit for $f = x_1 \oplus x_2x_3 \oplus x_2'x_3'$

vector is an all-zero vector. The second vector has only c_0 and c_1 as zero, while in the third vector only c_0 and c_2 are zero valued. The next vector consists of all 1's except for c_0. The next n vectors are made up of 'walking zero vectors'. Finally the last vector is an all-zero one except for the first element. The test matrix is the same for any logic function in Reed-Muller ESOP form provided that the number of data inputs is the same.

PROPOSED MODIFICATIONS AND SIMULATION RESULTS

Instead of the second and third vectors with 'walking zero vectors' (or '*weezee* vectors' to be short) fixed in columns c_1 and c_2, two modifications are now proposed: In one, which I prefer to call as 'AC weezee' method, all the control inputs participate in the 'zero walk'. The second method, which may be called as 'Alternative Vector Method' suggests c_2 and c_3 for 'weezee' instead of c_1 and c_2. The network structures and the test matrices for the reference method[15] as well as the now proposed modifications were simulated in matlab. The single stuck-at faults of type s-a-0 and s-a-1, at each of the data and control inputs, the literal-complementing xor gate outputs, the AND block outputs as also the function xor tree gate outputs were simulated by redefining the corresponding variable to zero or one. The test vectors were applied as the values of the simulated logic variables and the resulting outputs f, o1 and o2, each as a vector of n+5 elements (for the reference method and Alternative Vector method) or n+6 elements (for AC weezee method) were then converted to the equivalent decimal values and tabulated for convenience and easy comparison. The simulation results are as follows:

Example1: (n=3) $f = x_1 \oplus x_2x_3 \oplus x_2'x_3'$

f= Output of the function xor tree block

o1= Output of the separate AND gate

o2= Output of the separate OR gate

Reference method[15]

No fault => { f, o1, o2 }= { 118, 112, 127 }

Single stuck-at fault at one of control inputs c_0 to c_3, or data inputs x_1 to x_3:

Total number of possible faults: 7 x2= 14

Table 2: Decimal equivalents of the outputs for s-a-0 fault at each of input and control inputs for the reference method

	c_0	c_1	c_2	c_3	x_1	x_2	x_3
f	118	118	120	14	32	86	86
o1	112	112	112	112	0	0	0
o2	126	127	127	127	127	127	127

Table 3: Decimal equivalents of the outputs for s-a-1 fault at each of input and control inputs for the reference method

	c_0	c_1	c_2	c_3	x_1	x_2	x_3
f	46	118	119	118	126	118	118
O1	0	112	112	112	20	116	114
o2	255	127	127	127	255	255	255

Single stuck-at fault at one of intermediate gate outputs:

zl_1, zl_2: Outputs of literal-complementing xor gates; zl_1 equivalent to the complement x_2' and zl_2 for x_3' given in the function.

za_1, za_2, za_3: Outputs of the AND block gates; za_1 for the first product term x_1c_2, za_2 for the second term $x_2x_3c_3$ and za_3 for the last term $x_2'x_3'c_2$

zx_1, zx_2: Outputs of the function xor tree gates; zx_1 with za_1 and za_2 as inputs, while zx_2 (=f) with zx_1 and za_3 as inputs producing the final output.

Total number of possible faults: (2+3+2) x 2 = 14

Table 4: Decimal equivalents of the outputs for s-a-0 fault at each of gate outputs for the reference method

	zl_1	zl_2	za_1	za_2	za_3	zx_1	zx_2 (=f)
f	46	46	32	14	46	88	0
o1	0	0	112	112	112	112	112
o2	127	127	127	127	127	127	127

Table 5: Decimal equivalents of the outputs for s-a-1 fault at each of gate outputs for the reference method

	zl_1	zl_2	za_1	za_2	za_3	zx_1	zx_2 (=f)
f	114	116	223	241	209	167	255
o1	112	112	112	112	112	112	112
o2	255	255	127	127	127	127	127

Comments:

* Gross Total of possible single s-a-0 / s-a-1 faults: 14 + 14 = 28.
* Identical outputs for 'No fault' , sa0 / sa1 @ c1and

sa1 @ c3 { f= 118, o1= 112 and o2= 127}

→ 3 / 28 = 10.71% *completely unidentifiable*

* Same outputs for the following faults:

sa0 @ x_2 / x_3 {f, o1, o2}= {86, 0, 127 }

sa0 @ zl_1 / zl_2 {f, o1, o2}= {46, 0, 127 }

sa0 @ c3 / za_2 {f, o1, o2}= {14, 112, 127 }

→ 6 / 28 = 21.43 % *indistinguishable*

Proposed method: (Alternative vector method)

No fault => { f, o1, o2 }= { 86, 0, 127 }

Single stuck-at fault at one of control inputs c_0 to c_3, or data inputs x_1 to x_3: Total number of faults: 7 x2= 14

Table 6: Decimal equivalents of the outputs for s-a-0 fault at each of input and control inputs for the proposed method

	c_0	c_1	c_2	c_3	x_1	x_2	x_3
f	86	86	88	14	96	54	54
o1	0	0	0	0	0	0	0
o2	126	127	127	127	127	127	127

Table 7: Decimal equivalents of the outputs for s-a-1 fault at each of input and control inputs for the proposed method

	c_0	c_1	c_2	c_3	x_1	x_2	x_3
f	110	86	87	118	94	86	86
o1	16	0	0	0	0	0	0
o2	255	255	255	255	255	255	255

Single stuck-at fault at one of intermediate gate outputs: Total number of possible faults: (2+3+2) x 2 = 14

Table 8: Decimal equivalents of the outputs for s-a-0 fault at each of gate outputs for the proposed method

	zl_1	zl_2	za_1	za_2	za_3	zx_1	zx_2 =f
f	110	110	96	14	110	56	0
o1	0	0	0	0	0	0	0
o2	127	127	127	127	127	127	127

Table 9: Decimal equivalents of the outputs for s-a-1 fault at each of gate outputs for the proposed method

	zl_1	zl_2	za_1	za_2	za_3	zx_1	zx_2 (=f)
f	82	84	159	241	145	199	255
o1	0	0	0	0	0	0	0
o2	127	127	127	127	127	127	127

Comments:

* Gross total of possible single s-a-0 / s-a-1 faults: 14 + 14 = 28.

* Identical outputs for 'no fault', sa0 @ c1 { f, o1, o2} = { 86, 0, 127 }

→ 1 / 28 = 3.57 % *completely unidentifiable fault*

* Same outputs for

sa0 @ c4/za2 (14,0,127)

sa0 @ x1/za1 (96,0,127)

sa0 @ x2/x3 (54,0,127)

sa1 @ c2/x2/x3 (86,0,255)

sa0 @ zl1/zl2/za3 (110,0,127)

→ 12 / 28 = 42.86 % *indistinguishable*

Table 10: Comparison of simulation results for a few logic functions

Example No.	No. of Data inputs	Total faults	Reference Method[15]		Proposed Methods					
					Reference Method Vector		AC weezee Vector		Alternative Vector	
			%U	%I	%U	%I	%U	%I	%U	%I
1	3	28	10.71	21.43	3.57	46.43	3.57	42.86	3.57	42.86
2	3	28	10.71	21.43	3.57	50	3.57	46.43	3.57	46.43
3	3	30	7.14	20	7.14	50	7.14	42.86	7.14	42.86
4	4	32	9.38	21.88	3.13	43.75	3.13	40.63	3.13	40.63
5	5	38	2.63	23.68	NIL	55.26	NIL	44.74	NIL	44.74
6	6	38	10.53	28.95	2.63	52.63	2.63	50	2.63	50
7	7	40	7.5	35	2.5	47.5	2.5	42.5	2.5	45
8	8	54	NIL	35.19	NIL	44.44	NIL	44.44	NIL	44.44
9	9	52	1.92	34.62	NIL	46.15	Nil	46.15	NIL	48.08

U -- Unidentifiable I -- Indistinguishable

Example No. 1: f= x1 ⊕ x2x3 ⊕ x2'x3'

Example No. 2: f= x1 ⊕ x1x2x3 ⊕ x2'x3'

Example No. 3: f= x1' ⊕ x1'x2' ⊕ x2x3'

Example No. 4: f= x1x2x3 ⊕ x2x3x4 ⊕ x2'x3'x4'

Example No. 5: f= x1x5 ⊕ x1x2x3 ⊕ x2x3x4 ⊕ x2'x3'x4'

Example No. 6: f= x1x2x6' ⊕ x2x3x4 ⊕ x3'x4'x5'

Example No. 7: f= x1x2x7' ⊕ x3x4x5 ⊕ x4'x5'x6'

Example No. 8: f= x1x2x8' ⊕ x3x7'x6' ⊕ x4'x5' ⊕ x1'x2'x3'

Example No. 9: f= x1x2x8' ⊕ x3x7'x6' ⊕ x4'x5'x9 ⊕ x1'x2'x3'

A similar procedure is adopted for the reference method vector[15] as well as 'AC weezee' method, with proposed modified circuit. The results for the above as also a few additional examples are shown in Table 10.

CONCLUSION

Three test set schemes for detection of single stuck-at faults for logic functions have been proposed and the simulation results show that the proposed schemes reduce the possibility of unidentifiable faults. Further an additional index, the indistiguishability of faults, which is different from unidentifiability has also been proposed and compared for the example functions.

REFERENCES

1. Kalay, U., D.V. Hall and M.A. Petrowski, 2000. A minimal universal test set for self-test of EXOR-Sum-of-Products circuits. IEEE Trans. Computers, 49: 267-276.
2. Reddy, S.M., 1972. Easily testable realizations for logical functions. IEEE Trans. Computers, 21: 1183-1188.
3. Saluja, K.K. and S.M. Reddy, 1975. Fault detecting test set for Reed-Muller canonic networks. IEEE Trans. Computers, 24: 995-998.
4. Sasao, T., 1997. Easily testable realizations for Reed-Muller expressions. IEEE Trans. Computers, 21: 709-716.
5. Aborhey, S., 2001. Reed-Muller tree-based minimization of fixed polarity Reed-Muller expansions. IEE Proc. Comput. Digit. Tech., 148: 2.
6. Sasao, T. and P. Besslich, 1990. On the complexity of mod-2 sum PLAs. IEEE Trans. Computers, 39: 262-266.
7. Wu, H. *et al*., 1996. Generalized partially-mixed-polarity Reed-Muller expansion and its fast computation. IEEE Trans. Computer, 45: 1084-1088.
8. Das, D.K., S. Chakraborty and B.B. Bhattacharya, 2003. Universal and robust testability of stuck-open faults in Reed-Muller canonical cmos circuits. Intl. J. Electron., 90: 1-11.
9. Wu, H. *et al*., 1996. Generalized partially-mixed-polarity Reed-Muller expansion and its fast computation. IEEE Trans. Computer, 45: 1084-1088.
10. Drechshler, R. *et al*., 1997. Testability of 2 level AND/EXOR Circuits. Proc. European Design and Test Conf.
11. Pradhan, D.K., 1978. Universal test sets for multiple fault detection in AND-EXOR arrays. IEEE Trans. Computers, 27: 181-187.
12. Bhattacharya, B.B. *et al*., 1985. Testable design of RMC networks with universal tests for detecting stuck-at and bridging faults. IEE Proc., 132 Part E: 155-161.
13. Zhongliang, P., 2003. Bridging fault detections for testable realizations of logic functions. Proc. of Intl. Conf. on VLSI Design, pp: 423-427.
14. Rahaman, H., D.K. Das and B.B. Bhattacharya, 2004. Testing of stuck-open faults in generalised Reed Muller and EXOR sum of products cmos circuits. IEEE Proc. Comput. and Digit. Tech., 151: 1.
15. Zhongliang, P., 2002. Testable realizations of ESOP expressions of logic functions. Proc. of 11th Asian Test Symposium (ATS"02), IEEE Computer Society.