Generalized Reasoning Scheme for Redundancy Addition and Removal Logic Optimization

J. A. Espejo, L. Entrena, E. San Millán, E. Olías
Universidad Carlos III de Madrid
e-mail: {ppespejo, entrena, quique, olias}@ing.uc3m.es

Abstract*

In this work a generalization of the structural Redundancy Addition and Removal (RAR) logic optimization method is presented. New concepts based on the functional description of the nodes in the network are introduced to support this generalization. Necessary and sufficient conditions to identify all the possible structural expansions are given for the general case of multiple variable expansion. Basic nodes are no longer restricted to simple gates and can be any function of any size. With this generalization, an incremental mechanism to perform structural transformations involving any number of variables can be applied in a very efficient manner. Experimental results are presented that illustrate the efficiency of our scheme.

1. Introduction

Logic optimization methods based on the combined addition and removal of redundancies to a logic network have been proposed [1-5]. In all these methods, the basic optimization mechanism consists in adding redundant wires/gates so that other redundancies are created somewhere else, producing a smaller of faster network. These methods are usually applied on a structural representation of the circuit consisting of simple gates.

We generalize the concepts of redundancy removal and redundancy addition to structural reduction and expansion operations. Reduction is a generalized removal operation that consists in modifying an existing function in the way to reduce its support. Similarly, expansion is a generalized addition operation, that consists in modifying an existing function in the way to extend its support. With this generalization, expansion is a functional operation that is not limited to the addition of a single wire/gate. However, the efficient identification of multiple wires/gates for expansion is still an open problem because of the large number of possibilities that exist even in small circuits.

* This work has been supported by the Community of Madrid (Spain) under Project #07T/0007/1998

In the original work where Redundancy Addition and Removal (RAR) is proposed [1], the first step is the selection of a target wire for removal. Then a single wire expansion is computed to make this reduction feasible. An ATPG-based implication mechanism is used in order to check that expansions and reductions are valid. This approach has been followed in several other works [2][3][6]. In the Boolean Reasoning technique [4][5], a valid addition (expansion) is first identified by using recursive learning. Then, redundancy removal (reduction) is performed for the entire network. In both cases, there is not a strong link between the expansion and reduction operations. Instead, a trial and error technique is used to identify possible candidates. The consequence of this trial and error technique is that a high percentage of the computations are useless. Being m the number of possible candidates, up to m+1 tests are required to be computed in each iteration to explore the solution space. Many of the candidates are usually not redundant, resulting in a waste of time. Some techniques are used to reduce the number of candidates [1], but still a large number of them have to be tried. In this paper, enhanced operations will be presented demonstrating that it is possible to obtain both sufficient and necessary conditions to identify valid transformations.

Using the trial and error technique, the complexity grows exponentially when transformations involving the addition of several wires/gates are considered. Techniques for multiple wire expansion have only been proposed for some particular cases involving just two wires [3][8]. No general efficient solution has been proposed to identify transformations using multiple variable expansion. In this paper, we present a recursive method to compute all possible multiple variable expansions.

The remaining of this paper is as follows. Section 2 describes preliminary concepts for the generalization of the RAR technique to the enhanced operations. In section 3, enhanced expansion is presented based on functional representation of the nodes to be expanded. In section 5 a structural view for these transformations is given. Finally, experimental results and conclusions are presented.

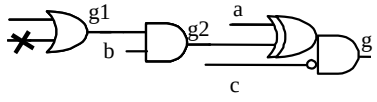
2. Redundancy generation

Our approach departs from the RAR technique described in [1]. In this technique, a wire w_r is selected for reduction and a redundancy test for fault f *s-a-'v'* is performed on this wire. Each path on which the fault effect is being propagated to an primary output (PO) is called a fault propagation path. A node is a fault dominator if all fault propagation paths share this node. Observability mandatory assignments (MAs) are binary values at internal nodes in the network obtained by assigning a sensitizing value ('1' for AND, '0' for OR) to side-inputs of dominators in the fault propagation path. A redundancy test consists in the implication of the observability MAs in order to obtain the complete set of MAs for fault f (f-SMA). If f-SMA is inconsistent, then the fault is redundant and some wires and gates can be directly removed by setting the tested wire to a constant binary value '*v*'. If f-SMA is consistent (f is testable), then an expansion is tried in order to make f redundant by blocking fault propagation.

The main objective of a expansion operation is to make the target fault f unobservable. This is performed by taking into account the observability conditions for the fault f. Observability conditions do exist for each node that generates and inconsistency if the node is assigned to a particular value. Note that these include fault dominator nodes as well as some other nodes in the transitive fanin of a dominator. Observability conditions then generate MAs. We present now how to compute the observability conditions by using node's truth tables and how observability MAs can be also calculated with this information.

Let's consider the truth table of a fault dominator node in the network. Consider also the truth table variables are ordered in such a way that the fault propagating input (P) is set in the Least Significant Bit (LSB) position. Each row in the table will be refered as a *term*. Two P-adjacent terms of the function will be referred as a *pair*. With this representation, the node´s output values give the necessary information to compute the fault observability conditions. If the node's output values for a pair are the same for both terms in the pair (i.e. both '0' or both '1'), then there is an observability don't-care (ODC) for any value of P input for this pair, i.e, no observability conditions can be computed for this pair. If the terms in a pair have different output values, then the fault propagating at P input is observable and then there exist observability conditions for this pair. Mandatory assignments can be calculated as the common values for the side inputs.

Example 1. For the circuit in Fig. 1, a fault f is propagating through dominator nodes g2 and g3. Truth tables for nodes g2 and g3 are also shown. For the simple gate g2, the pair T0-T1 does not infer any fault observability condition, as in both terms the output value is the same ('0'). For the pair T2- T3, the observability assignment b=1 is inferred, as the output value is different for this pair. To show how observability conditions are a more general view than assignments, consider now node g3. In this case there are two observability MAs: b=1, c=0 for pair T0-T1; and b=1,c=0 for pair T4-T5. The assignments for these two pairs can be merged and the only final mandatory assignment for this function is c=0. Note that the observability condition concept is more complete in terms of observability information of a fault, as two fault observability conditions are identified for node g3, although only one single assignment is inferred.



g2 truth table

	b	g1	g2
T0	0	0	0
T1	0	1	0
T2	1	0	0
T3	1	1	1

g3 truth table

	a	c	g2	g3
T0	0	0	0	0
T1	0	0	1	1
T2	0	1	0	0
T3	0	1	1	0
T4	1	0	0	1
T5	1	0	1	0
T6	1	1	0	0
T7	1	1	1	0

Fig. 1. Example of observability conditions

A expansion operation consists in a modification of the function performed by the node in order to set the same output value to both terms in the pair. This way a local discrepancy between the original and the modified dominator function is generated in one term in each pair with observability conditions. These pairs are called *necessary discrepant pairs*, as a discrepancy is necessary on each of these pairs to block fault propagation. If this function modification is unobservable at any PO, then the reduction of the fault can be directly performed as the overall functionality of the circuit is not changed. The unobservability of the change is only obtained if the discrepant term exactly fits within an existing or newly created local don't-care (LDC) of the circuit.

LDCs can be identified by implication techniques. If the initial LDC set does not include ODCs at discrepant pairs, new LDCs should be generated at the discrepant pairs in the circuit by means of a expansion operation. If such as a expansion operation can be performed, then the target wire fault becomes redundant.

3. Enhanced expansion

A expansion operation must not alter the overall functionality of the circuit. Therefore, the discrepancies introduced by a expansion operation must be LDCs. To know if the expanded discrepant terms are LDCs, an implication scheme can be adopted starting from the

observability conditions of the discrepant terms. The following lemma models this implication scheme.

Lemma 1.- The conditions for the discrepant terms at the expanded node to be LDCs can be modeled as the redundancy test of a new fault f2 s-a-'u' at input P of the expanded node. The value of 'u' will depend on the MA at P for the discrepant terms. For P='0' at the discrepant term value for u is '1'. For P='1' at the discrepant term value for 'u' is '0'.

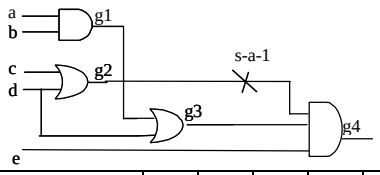
The following theorems provide the means to identify transformations involving single and multiple variable expansion, respectively.

Theorem 1. Let f_1 be a fault being tested at wire w_r. Let gd be a node at which there are observability conditions for fault f_1. Let f2 be the fault at input P of gd, defined in Lemma 1. Let gn be a node that has MAs 'v1' and 'v2' for the faults f1 and f2, respectively. The necessary and sufficient condition for node gn to be a single wire alternative node is that 'v1' and 'v2' exist and *'v1'= NOT 'v2'*.

Theorem 2. Let gn1 be a node that has a mandatory assignment 'v' for fault f1 (f2) and no mandatory assignment ('U') for the other fault f2 (f1). If there is a node gn2 that meets the conditions of Theorem 1 when f2-SMA (f1-SMA) is extended with the assignment gn1 = 'v', then a transformation involving a two variable expansion with nodes gn1 and gn2 is possible.

The proof of these theorems is omitted because of lack of space. Instead, we will illustrate the application of the theorem with examples. A partial demonstration for the case the expanded node is a dominator is given in [9].

Example 2.- In the circuit in Fig. 2, fault g2-g4 s-a-1 is requested for reduction. This fault is tested and implication results are also shown in in Fig. 2 (U value means "unknown") . Node g1 at the transitive fanin of g4 dominator is selected for expansion. General expansion scheme of g1 is shown in Fig. 3. After expansion, module g1 has three inputs: a,b and the new input N.



	g1	g2	g3	g4	a	b	c	d	e
g2->g4 s-a-1	1	0	1	0/1	1	1	0	0	1
b->g1 s-a-0	1	1	0	U	1	1	1	0	U

Fig 2. Single variable transformation at dominator´s transitive fanin

To generate an ODC for the faults an inconsistency in the f-SMA is inserted. The forced value for g1 is 1 when fault f is tested. Then, to generate f-SMA inconsistency, g1 must generate the value '0' after expansion. This is must be performed for all PI combinations that test the fault, with assignments a=1,b=1. Discrepancy between the original function and the extended function will be located at term T3 or term T7. To compute if T3 (T7) is a LDC, an implication procedure is used. Initial values are a=1,b=1 and the observability assignments of these values, which is equivalent to the test of a new fault f2 b->g1 s-a-0. If after the f2 implication an assignment is computed in some node, explicit information about LDC can be inferred. In this example f2 test imply c=1. Because of that we conclude that N(c)=0 a=1 b=1 is a LDC under our test conditions. This is, it is impossible to occur the observable combination a=1,b=1,c=0 and because of that discrepant term is T3 and new node for expansion is c. Note that extended function can be Shanon decomposed by means of new variable N. The part with the assignment N=0 will represent all the input combinations able to test the fault. Associated cofactor will be named sensitive Cofactor (Cs). Input combinations unable to test the fault share the MA N=1. Associated cofactor will be named non sensitive cofactor (Cns).

		N(c)	a	b	g1*
Cs	T0	0	0	0	0
	T1	0	0	1	0
	T2	0	1	0	0
	T3	0	1	1	***0***
Cns	T4	1	0	0	0
	T5	1	0	1	0
	T6	1	1	0	0
	T7	1	1	1	1

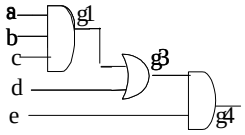


Fig 3. Single variable transformation scheme and results

This extended formulation allows to formulate the multiple addition expansion as the generalization of the described technique. Main advantage of this generalization is that an incremental implication scheme can be used. The following two examples will illustrate this idea.

Example 3.- In the circuit in Fig. 4, f1 d->g6 s-a-1 fault is selected as the target fault. For our transformation purposes, dominator node g9 is selected for expansion. After an implication of f1 we see that g9 value at non faulty circuit is '0'. Because of that, selected value to generate the LDC is '0'. T3 is the proposed discrepant term in this case and we select f2 to be g8,g9 s-a-0. Implications are shown in Fig. 4.

As no single alternative node can be found, we try to include more observability conditions at the discrepant pairs. Then, we focus on a node that has a mandatory assignment for at least one of the faults, such as g1. We are not able to determine if g1 value by itself will generate an adequate LDC, as its implied value is 'U'. We set g1='0' (equal value than the f1-MA) and repeat f2 test with this extended value. After implication, we find that

the inexistent MA g2='U' is converted in g2='1' for the extended set. In this case, under the initial observability conditions it is impossible to obtain combination g1='0' and g2='0'. Because of that, new LDC have been successfully generated at the cofactor with g1='0' and g2='0' when this double variable expansion is performed.. In this case reduction operation yields to the elimination of nodes g6 and g7. Expansion for g9 is g8 f (g1+g2).

	g1	*g2*	*g3*	*g4*	*g6*	*g7*	*g8*	*g9*	*a*	*b*	*c*	*d*	*e*	*f*
f1	0	0	1	0	0/1	0/1	0/1	0/1	1	1	U	0	U	1
f2	U	U	U	U	U	U	1	1/0	U	U	U	U	U	1

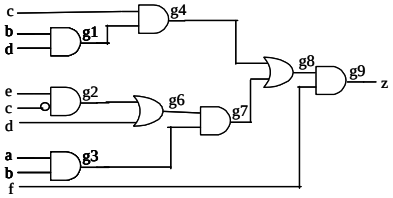
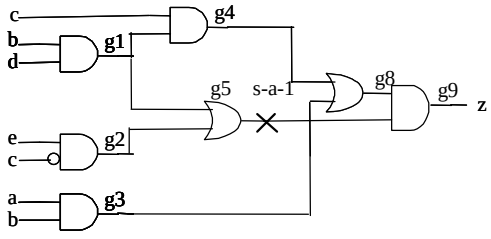
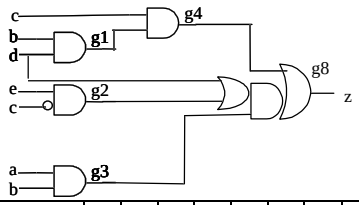


Fig 4. Multiple variable transformation at the dominator

Example 4.- In the circuit in Fig. 5, fault g5-g9 s-a-1 is requested for reduction. Implications for this fault are also shown in Fig. 5. Node g8 is considered for expansion. As g8 is located in the transitive fanin of g9 dominator, the objective is to generate an inconsistency in the forced f-SMA The forced value for g8 is 1 when fault f is tested. To generate f-SMA inconsistency, g8 must generate the value '0' after expansion when the fault is being tested, this is, when g4=0, g3=1 . Initial values for implication are g4=0, g3=1 and the observability assignments of these values. In this case this is modeled as the f2 fault test g3-g8 s-a-0. After this second implication, it can be seen that no candidate is found for single wire addition. Then, we focus on a node that has a mandatory assignment for at least one of the faults, such as g2. We set g2='0' and repeat f2 test with this new value (implication3). After implication3, we find that the MA d='1'. Under the initial observability conditions it is impossible to obtain combination g2='0' and d='0'. Because of that, new LDC have been successfully generated at the cofactor with g2='0' and d='0' when this double variable expansion is performed. Final circuit is shown in Fig. 5.





	g1	g2	g3	g4	g5	g6	g7	a	b	c	d	e
g5-g9 s-a-1	0	0	1	0	0	0/1	0/1	1	1	U	0	U
g3-g8 s-a-0	U	U	**1**	**0**	**1**	0	0	1	1	U	U	U
implication3	1	**0**	**1**	**0**	**1**	0	0	1	1	0	1	1

Fig 5. Multiple variable transformation example

4. Expansion module

The previous generalized scheme has been expressed in terms of functional description of a node. In this section we will see that the expanded node can be calculated in all cases from the original one by adding an expansion module input at which fault is blocked. With this approach, the functional representation of the node is no longer needed and the whole process can be performed with structural techniques.

Lemma 2.- Let 'u' be the binary value calculated according to Lemma 1 when an expansion is computed. Output value of both terms in each pair on Cs can be always calculated as the value of each term in the particularization of the original node function for P='u'.

Example 5 - In Example 2, if g1 is particularized for a=0 or b=0, we obtain a function with a pair PT0 with value '0' and pair PT1 with value '0'. We compute Cs terms T0 and T1 output value as PT0 output value (non discrepant pair), and terms T2 and T3 at the discrepant pair as PT1 output value.

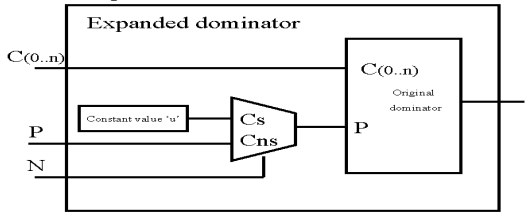


Fig 6. Expanded dominator

Expanded function can be always computed for both Cs and Cns by setting input P to a constant value for Cs and setting input P to its original value for Cns. The general form of the expansion module is a multiplexer, as described in Fig. 6. By particularizing the module for the different binary values of N_i and P, all families of transformations can be obtained. For a single variable expansion, a four member family of transformations is obtained by particularizing expansion for values 'u' and 'Cs'. For two-variable expansion, an eight-member family of transformations is found, and so on. Note that with our method we know a priori which of these transformations must be applied in each case without the need to try everyone of them.

5. Experimental results

In this section, we provide experimental results for benchmark combinational circuits. Our goal is to show the improvements in the context of previously proposed optimization algorithms. To this purpose, we modified RAMBO tool [1] to work with the enhanced algorithm. The experiments were performed in a Sun Ultra 1 WS.

Original benchmarks were first optimized with script.rugged included in SIS[7] After this optimization, both the enhanced and the original algorithm were run in parallel over the same initial optimized circuits. Both algorithms gave the same optimization results, but the enhanced algorithm found these transformations in a 38% less of time in average. Table 1 summarizes the results of the comparison between the old and enhanced algorithms regarding efficiency issues. In this table, columns 2 and 3 show the CPU time used to perform optimization with the old (OLD) and the enhanced algorithm (ENH). Columns 4 and 5 show the number of total test performed in both cases. Columns 6 and 7 show the number of considered alternative nodes also in both cases. In all circuits, a significant reduction of CPU time is obtained. The number of total tests performed in the optimization has been drastically reduced due to the elimination of the old trial and error search for alternative nodes. This reduction exceeds 90% in average. The decrease in the number of alternative nodes considered to perform reduction is also shown. Results show that for these pre-optimized circuits, only 0.2 % of the candidates were really alternative nodes useful to perform area optimization in the old algorithm.

In Table 2, the results using addition of two variables are compared against the algorithm used in [8] for timing optimization. In this case, the reduction in CPU time (83%) and number of tests (92%) was much higher, showing clearly the advantage of the techniques proposed in this work.

6 . Conclusions and future work

We have presented a generalization of the structural logic optimization methods (RAR). This generalization is based on functional considerations, but can be applied in a purely structural manner. Transformations involving multiple variable expansion can be identified very efficiently by performing additional implications over two initial tests, overcoming the previous approaches that were limited to single or two variable expansion.

7. References

[1] K.-T. Cheng, L. Entrena. “Sequential Logic Optimization by Redundancy Addition and Removal”. Proc. ICCAD’93

[2] S.-C. Chang, K.-T. Cheng, N.-S. Woo, M. Marek-Sadowska. “Post-layout logic restructuring using alternative wires”. IEEE Transactions on CAD, vol.16, n. 6,

[3] S. C. Chang, M. Marek-Sadowska, K.-T. Cheng. “Perturb and Simplify: Multilevel Boolean Network Optimizer”. IEEE Transactions on CAD, vol. 15, nº 12

[4] W. Kunz, P. Menon. “Multi-Level Logic Optimization by Implication Analysis”. Proc. ICCAD-94, p.6-13. Nov.1994

[5] W. Kunz, D. Stoffel. “Reasoning in Boolean Networks: logic synthesis and verification using testing techniques”. Ed. Kluwer Academic Publishers, 1997

[6] S. C. Chang, L. P. van Ginnekan, M. Marek-Sadowska. “Fast Boolean Optimization by Rewiring”.Proc. ICCAD’96.

[7] “SIS: A System for Sequential Circuit Synthesis” Report M92/41, University of California, Berkeley, May. 1992.

[8] L. Entrena, J. A. Espejo, E. Olías, J. Uceda. “Timing optimization by and Improved Redundancy Addition and Removal Technique”. Proc. EURO-DAC’96,

[9] J. A. Espejo, L. Entrena, E. San Millán, E. Olías. “Functional extension for structural logic optimization techniques”. Proc. ASP-DAC’01. January, 2001.

	CPU (s)		#TESTS		ALT. NODES	
	OLD	ENH	OLD	ENH	OLD	ENH
alu2	14	5	13953	1305	47482	177
alu4	110	46	54635	2385	173261	382
apex6	27	24	38224	2855	82606	175
C1908	9	8	10423	1813	24072	31
C3540	369	202	147181	18070	534355	907
C5315	58	51	44848	6251	93891	434
C7552	310	236	128289	17212	294463	1358
frg2	113	56	92928	3257	216757	648
i6	58	52	55645	1610	139551	3
i7	118	88	89914	2037	249753	72
i8	334	177	189187	4143	636336	1001
k2	321	182	151545	2821	525216	628
pair	58	46	52889	5844	114494	486
vda	41	26	32858	1287	104936	271
x3	31	25	39251	3643	84612	199
TOTAL	1971	1224	1141770	74533	3321785	6772

Table 1 Experimental Results for single wire expansion

	CPU (s)		TOTAL TESTS		ALTERNAT.	
	OLD	ENH	OLD	ENH	OLD	ENH
alu2	65	8	11902	670	682750	347
alu4	198	52	36313	1407	5589977	817
apex6	15	4	3520	418	98653	1319
C1908	377	82	13322	1245	28908	71
C3540	299	30	14830	891	1861563	372
C5315	69	29	20732	1367	655746	1146
C7552	1218	226	113122	9398	1732412	1352
frg2	19	6	5463	334	478876	989
i6	2	0	201	38	181	4
i7	2	0	319	49	269	4
i8	9	8	3089	352	224905	183
k2	988	68	33725	1036	6176786	23062
pair	105	37	22369	2763	500583	642
vda	120	19	14800	644	1861213	3371
x3	17	1	3356	333	87030	70
TOTAL	3503	570	297063	20945	19979852	33749

Table 2 Experimental Results for multiple wire expansion