Functional extension of structural logic optimization techniques

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 - This work provides a generalization of structural logic optimization methods to general boolean networks. With this generalization, the nodes of the network are no longer restricted to simple gates and can be functions of any size. Within this framework, we present necessary and sufficient conditions to identify all the possible functional expansions of a node that allow to eliminate a wire elsewhere in the network. These conditions are also given for the case of multiple variable expansion, providing an incremental mechanism to perform functional transformations involving any number of variables that can be applied in a very efficient manner. On the other hand, we will show in this paper that relevant simplifications can be obtained when this framework is applied to the particular case of AND-OR-NOT networks, resulting in important savings in the computational effort. When compared to previous approaches, the experimental results show an important reduction in the number of computations required.

I. INTRODUCTION

During the last years, new logic optimization methods have been proposed that are based on ATPG concepts and techniques [1-8]. Contrast to the more traditional methods based on techniques to identify common sub expressions, ATPG-based methods exploit the concept of redundancy in a direct manner and perform logic optimization by an iterative addition and removal of redundancies. These methods have demonstrated to produce excellent results, being the major advantages the low memory usage and the short run times. On the other hand, they are suitable for post-mapping and post-layout optimization, as they can work satisfactorily under technological constraints [3][7].

Redundancy Addition and Removal (RAR) methods are applied on a pure structural representation of the circuit, as this representation facilitates the required implication process [1][6]. For this reason, they are sometimes called *structural methods*. The only functional information on these representations is implicit in the use of logical AND-OR-NOT gates representing basic boolean operations. The rest of the information is given by the interconnection of these basic functional units. Even in the functional application of RAR to post-layout logic optimization of LUT-based FPGAs, the function performed by each LUT is converted to an AND-OR-NOT representation and the subsequent reasoning is based on the properties of these basic boolean functions [3]. Therefore, and AND-OR-NOT decomposition has to be performed before applying RAR methods.

In this work a generalization of the RAR methods based on the functional description of some of the nodes of the network is presented. With this generalization, the nodes are no longer restricted to simple gates and can be any function of any size. This provides the theoretical framework for new relevant conclusions to be found. The result is a general enhanced technique with important savings in the required computational effort. This enhanced technique is open to a new range of future applications, particularly including post-mapping and post-layout logic optimization.

Complexity is a very important issue in all RAR approaches. Usually, only a very small percentage of possible wires is redundant and even a smaller percentage of them would create other redundancies in order to produce a useful transformation. The complexity grows exponentially when transformations involving the addition of several wires/gates are considered. Several approaches have been proposed to tackle this problem. In the original work where RAR is first proposed [1], a target for removal is first selected and a redundancy test is performed for the target. Addition is only attempted from the nodes that obtain a mandatory assignment for this test. However, many of the addition candidates obtained this way are usually not redundant and cannot be added, as this is only a sufficient condition. This approach has been followed in several other works [3][4][9]. In the Boolean Reasoning technique [5][6], a valid addition is first identified by using recursive learning. Then, redundancy removal is performed for the entire network. However, there are many additions that generate no new redundancies and therefore do not contribute to any optimization. In summary, both approaches start by identifying a target for removal or addition, and then try all possible candidates in order to identify a valid transformation. The consequence of this "trial and error" procedure is that a high percentage of the computations are useless. Being m the number of candidates, up to $m+1$ tests are required to be computed in each iteration. Some heuristics and fault dominance conditions have been developed to accelerate this search. It will be demonstrated that the number of computations can be greatly reduced by obtaining both sufficient and necessary conditions for valid and useful transformations. In particular, for a given destination node these conditions can be evaluated by performing only two redundancy tests. Moreover, with the extension of the method to general functions, transformations involving the addition of multiple wires can also be identified very efficiently by performing additional implications over these two tests.

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

II. OBSERVABILITY CONDITIONS

Let w_r be a wire for removal in a boolean network. To check whether the wire w_r is redundant, a test for *stuck-at-'v'* fault *(f s-a-'v')* is performed on the network. Redundancy test is based on the implication of *the set of mandatory assignments for fault f* (f-SMA) [1]. *Observability mandatory assignments* are obtained by assigning a sensitizing value to side-inputs of *absolute dominators* [1] in the fault propagation path. If the f-SMA is inconsistent, then the fault is redundant and the associated wires and gates can be directly removed by setting w_r to a constant binary value 'v'.

If f-SMA is consistent (f is testable), RAR method tries to block fault propagation to any primary output (PO) at internal nodes in order to generate a new redundancy in w_r. Blocking of fault propagation at one of the fault dominators is the key for generating redundancies in the circuit. It can be achieved in two ways, depending on the location of the node at which fault propagation is blocked. The first way is the addition of a set of wires/gates to an absolute fault dominator so that, for all PI combinations that test the fault f, the inserted wires/gates prevent fault propagation[2]. The second way is the addition of a set of wires/gates to a gate located in the transitive fanin of a dominator so that, for all PI combinations that test the fault f, the inserted wires/gates generate an inconsistency in f-SMA[4]. All the results presented in this paper apply to both cases. However, for the sake of brevity, we will present only demonstrations for the first case. The term *expansion* is used for a generalized addition operation, that consists in modifying an existing function in the way to extend its support. Similarly, the term *reduction* is used for a generalized removal operation that consists in modifying an existing function in the way to reduce its support.

Our formulation for the computation of the *observability assignments* focuses in the dominator´s truth table. This formulation is best understood if 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. This way all pairs of terms adjacent by input P are also adjacent in the table. It must be noted that, with this representation, function values for each pair indicates the fault observability assignments for the fault. If the dominator's output value for a pair is the same for both terms in the pair (i.e. both '0' or both '1') then for this pair there is an observability don't-care (ODC), as the effect of a fault propagating through P is unobservable in this pair. If the terms in a pair have different output values, then the effect of P is observable and there are observability assignments for this pair.

Example 1.- In the case of a two-input AND gate with inputs P (fault propagating input) and S (side input), the pair of terms T0 (S=0, P=0) and T1 (S=0, P=1) do not infer any observability assignment, as in both terms the output value is set to the same value '0'. For the second pair of terms, T2 (S=1, P=0) and T3 (S=1, P=1), the observability assignment S=1 is inferred for the fault propagating trough P, as the AND output value is different for this pair.

III. REDUNDANCY GENERATION

Redundancy can be generated by considering observability don´t cares (ODC) at each pair having observability conditions. In order to block fault propagation, the dominator´s function is modified to set the same output value to both terms in the pair. This way a discrepancy between the original and the modified dominator´s function is generated in one term of each observable pair. These pairs are called *necessary discrepant pairs*, as discrepancy is necessary at these pairs in order to block fault propagation. To generate new ODC, all necessary discrepant pairs at the dominator must be set to the same value.

If this function modification is unobservable at primary outputs (PO), then the reduction of the fault can be directly performed as the overall functionality of the network is not changed. Also, if this function modification is observable at PO, then an error has been introduced in the network, and an expansion operation must be performed at the dominator in order to compensate it.

If the modified discrepant term exactly fits within an existing local don't-care (LDC), the change in the dominator´s functionality can be directly performed without generating error in the network . Then, an implication-based LDC search can be performed to detect the initial LDC set at terms in the dominator. 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 an expansion operation.

IV. ENHANCED SINGLE VARIABLE EXPANSION

In this section we present the fundamental lemmas and theorems that provide the necessary and sufficient conditions for transformations based on single variable expansion. Suppose a fault f is tested in an irredundant target wire w_r selected for removal. Let consider fault dominator node g_d to perform expansion. Expansion is performed by adding a new variable N to g_d from alternative node g_n. Node g_n has a mandatory assignment 't' when fault f is tested.

Generation of new ODC .- For fault *f*, all primary input (PI) combinations of the circuit can be divided in two sets. The first set is characterized by the assignment g_n*=NOT('t')*. Since the opposite mandatory assignment is required to test f, we can ensure that f is not testable for any of the PI combinations in this set. We call this *the non sensitive set.* The second set, called the *sensitive set*, is characterized by the assignment g_n*='t'*. All PI combinations that test the fault *f* belong to the sensitive set. Note that only a subset of PI combinations in the sensitive set makes the fault observable. This is because all PI combinations that can test the fault require g_n*='t'*, but not all PI combinations that make g_n*='t'* can test the fault.

Expansion operation produces expanded terms at the dominator´s truth table. The number of bits of each term is incremented in one and the number of terms in the

dominator's truth table is multiplied by two. These expanded terms can be associated to the Shannon cofactors [6] of the expanded function with respect to the added variable *N*. Only one of these cofactors, called the *sensitive cofactor*, will be *sensitive* to the fault test. Because of the above mentioned reasons, only some terms in the *sensitive cofactor* (C_s) need to be modified. To generate ODC for fault *f*, at fault-sensitive PI combinations of the sensitive set, function modification is performed only at dominator terms fulfilling two conditions: (1) the term belongs to *Cs*, and (2) the term belongs to a necessary discrepant pair. For all terms at the *non sensitive cofactor* (C_{ns}) the fault is already unobservable. This way, C_{ns} is made equal to the original function as for the extended terms in this cofactor no pair has observability conditions.

Example 2.- For circuit in fig 1, a reduction on the functionality of g_1 is proposed by applying a constant value 1 at the wire w_r = *c-g1*. To this purpose fault f is (*c-g1,s-a-1*). The f-SMA for this fault is shown in table 1. As the f-SMA is consistent, this fault is not redundant and therefore the reduction cannot be directly performed. g_4 is an absolute dominator in the fault propagation path. Then the expansion of g_d=g_4 is considered. To calculate the expanded function we propose cofactors of the expanded function with respect to a new variable *N*. General expansion scheme of g_4 is shown in table 2. After expansion, extended module g_4 has three inputs: the input through which the fault is propagated, P (node *g3*); the new input that expands the module, N (node g_n, unknown for the moment); and a side input S (node *e*). The non sensitive cofactor is then constructed by setting C_{ns}=g_4. For the calculation of C_s, an ODC must be generated at input P for each discrepant pair at this cofactor. For this example, the only necessary discrepant pair is located at terms T2-T3. To generate the ODC we set in both terms the same binary value *a*. To calculate value of *a*, and to calculate node g_n conclusions of next lemmas and theorem are used.

Lemma 1.- *"In AND-OR-NOT circuits, C_s can be always obtained from the original functionality of the dominator by simply modifying the function at one single term."*

Demonstration.- For AND-OR-NOT networks, the number of discrepant pairs at the dominator is always equal to one in all cases. This is because the cardinality of the off-set (on-set) is always one if the dominator is an OR (AND) gate. For this reason, it is only necessary to modify the function at one single term (discrepant term) to make both values at the unique discrepant pair equal to each other.

If the target wire fault f propagates a composite value 1/0 or 0/1 at the dominator´s output, the discrepant term can be directly identified according to lemma 2

Lemma 2.- *"Let COMP be a composite value obtained at the output of an unexpanded dominator as a result of testing fault f. The discrepant terms at C_s are the terms with output value '1' when COMP='0/1'. Also, the discrepant terms at C_s are the terms with output value '0' when COMP='1/0'."*

Demonstration.- Expansion must correct the error generated when the reduction of the target wire w_r is performed. This error can be modelled with the composite value at the dominator node. The correction of the original value involves generation of *not faulty circuit* value at the expanded functionality for the discrepant pair. If COMP='0/1' (not faulty circuit value='0') then both terms at discrepant pair must be set to '0'. Therefore, the terms with output value '1' must be the discrepant terms. The demonstration is similar for the case COMP='1/0'.

Expansion operation must not alter the overall functionality of the circuit. To this purpose, the expansion is only valid if new LDC are also generated at modified terms in the dominator when the expansion is performed.

Generation of new LDC.- To know if the effect of a expanded discrepant term can be observed at any PO, i.e., if the expanded discrepant term is an LDC, an implication scheme can be adopted. In this scheme, both justification and propagation assignments for this term are used as initial values for implication. Justification assignments for this term are the values of the discrepant term at the dominator´s inputs. Propagation assignments for this combination are all observability assignments at the nodes in the circuit necessary to observe the dominator's value at any PO under these inputs. It is known that if these assignments cannot be consistently implied, then there is an LDC at the discrepant term.

Lemma 3.- *"The described implication scheme can be modelled in all cases as the test of a new fault f2 s-a-'u' at dominator`s input P."* The value of 'u' will depend on the value of P for the discrepant term. For P='0' at the discrepant term, the value for u is '1'. For P='1' at the discrepant term, the value for u is '0'.

Demonstration.- As the discrepant term belongs to a discrepant pair, all necessary observability conditions of any fault propagating at P input are set by the values of the variables at the discrepant term. Therefore, the justification assignments for the discrepant combination coincide with the observability mandatory assignments of *f2* in this dominator. Also, *f2* is propagated from the considered dominator. Because of that, the fault propagation assignments are the same. Note that if the discrepant terms cannot be directly identified, value of '*u*' is not known, and both tests f2-*s-a-0* and f2-*s-a-1* must be performed.

Corollary. The location of new required LDCs can be performed by redundancy testing of the fault f2 *s-a-'u'* test at the fault propagating input in the dominator.

If the test fails for f2 *s-a-'u'*, it can be deduced that a new LDC has been generated as the discrepancy is unobservable. But also, if the implication does not fail, the derived mandatory assignments generated by this new implication can be used to find valid alternative nodes. The mandatory assignments obtained by testing the fault f2 at the unexpanded dominator provide all possible alternative expansions in this dominator. This way, the trial and error technique used in

previous works is eliminated as transformations are a priori known to be valid with the single test of f2. The fundamental theorem that validates this statement is provided next.

Let g_n, g_d and g_r be nodes in a boolean network. Let f1 be a s-a-'v' fault being tested at node g_r. Node g_d is an absolute dominator in the fault propagation path. It will be expanded with new variable N from g_n. Let *P* be the fault propagating input at node g_d. Let 't' be the mandatory assignment at node g_n when f1 is tested. Let 's' be the mandatory assignment at node g_n when a stuck fault f2 *s-a-'u'* is tested at input P.

Lemma 4.- *"If f1 is tested to be irredundant, f2 will always be tested as irredundant"*

Demonstration.- If f2 is redundant, f1 has to be redundant. If f2 is redundant, then a direct reduction of the fault could be performed. This reduction will block the f1 fault propagation path, making f1 redundant as well.

Theorem 1. The necessary and sufficient condition for g_n to be a single wire alternative node is that 't' and 's' exist and *'t'= NOT 's'*.

Demonstration of sufficient condition.- Since g_n has a MA 't' for f1, then N='t' will be the sensitive cofactor. All necessary discrepant terms to block f1 will be those pairs with observability conditions at the sensitive cofactor. On the other hand, the observability conditions at node g_d are the same for both faults f1 and f2. If node g_n has MA 's' for f2 we conclude that implying the observability conditions of f2 will never set value 't' = NOT 's' at node g_n. Therefore, the assignment N = 't' plus the observability assignments at node g_d is a LDC. This LDC is located exactly in one of the terms of each necessary discrepant pair. If f2 is a s-a-1 test, terms at discrepant pairs with P='0' are new LDC. If f2 is a s-a-0 test, new LDC are generated at each discrepant pair in terms with P='1'. This is, expanded combinations always generate a don´t care in one term of every discrepant pair in the sensitive cofactor. Because of that, the dominator`s output value at this term can be changed without affecting the overall functionality of the circuit.

Demonstration of necessary condition.- If 's' does not exist, no LDCs can be generated at the discrepant pairs. Therefore, it is not possible to modify the function to eliminate all the discrepancies, and g_n is not an alternative node. If 't'= 's', the new LDCs are generated in the non sensitive cofactor, and the discrepant pairs in the sensitive cofactor cannot be eliminated. Finally if 't' does not exist, i.e., no mandatory assignment is obtained at N for the fault f1, there are test vectors that set g_n='1' and test vectors that set g_n='0'. Then there are discrepant pairs in the two cofactors with respect to N. Since f2 is not redundant, the new LDCs can only be generated at one cofactor. Therefore, it is not possible to block the fault f1 at both cofactors.

Corollary. All possible transformations involving single wire expansion for a given dominator are obtained by applying Theorem 1.

Example 2 (contd.) Compute value for *'a'* and node g_n *in example 2. Va*lue at the good circuit when fault f is tested is '1/0'. This way *'a'* must be set to '1' and the only discrepant term is identified to be T2. Note that all the PI combinations that test the fault *c-g1* stuck-at-1 must imply *N = 't'*. For the input combinations that make *N = NOT 't'* the fault is already unobservable (C_{ns}). Node g_n must be selected to generate new LDC. For g_n to be an alternative node it must have a mandatory assignment *'t'* when the *c-g1* s-a-1 fault is tested and 'not t' when f2 is tested (Theorem 1). The f2 stuck value is selected considering that value of P at T2 is '0' (Lemma 2). For this case, f2 g3->g4 s-a-1 is tested. Results of these implications are also shown in table 1. (U value means "unknown")

TABLE 1. IMPLICATIONS FOR EXAMPLE 2

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

TABLE 2. EXPANSION SCHEME

		N	S	P	g4*
Cs	T0	't'	0	0	***0***
	T1	't'	0	1	***0***
	T2	't'	1	0	***'a'***
	T3	't'	1	1	***'a'***
C_{ns}	T4	NOT 't'	0	0	***0***
	T5	NOT 't'	0	1	***0***
	T6	NOT 't'	1	0	***0***
	T7	NOT 't'	1	1	***1***

TABLE 3. PERFORMED EXPANSION

		N	S	P	g4*
Cs	T0	0	0	0	***0***
	T1	0	0	1	***0***
	T2	0	1	0	***1***
	T3	0	1	1	***1***
C_{ns}	T4	1	0	0	***0***
	T5	1	0	1	***0***
	T6	1	1	0	***0***
	T7	1	1	1	***1***

TABLE 5. EXPANSION SCHEME FOR EXAMPLE 3

		N1	N2	S	P	g9*
Cs	T0	0	0	0	0	***0***
	T1	0	0	0	1	***0***
	T2	0	0	1	0	***'a'***
	T3	0	0	1	1	***'a'***
C_{ns}	T4	0	1	0	0	***0***
	T5	0	1	0	1	***0***
	T6	0	1	1	0	***0***
	T7	0	1	1	1	***1***
C_{ns}	T8	1	0	0	0	***0***
	T9	1	0	0	1	***0***
	T10	1	0	1	0	***0***
	T11	1	0	1	1	***1***
C_{ns}	T12	1	1	0	0	***0***
	T13	1	1	0	1	***0***
	T14	1	1	1	0	***0***
	T15	1	1	1	1	***1***

TABLE 4 IMPLICATIONS FOR EXAMPLE 3

	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

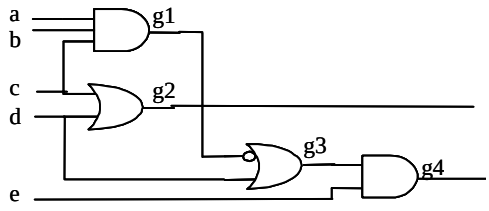

Fig 1. Example for single wire expansion.

Note that different values are obtained for g3->g4 s-a-1 and for c->g1 s-a-1 at nodes g_2 and c. Because of that, both nodes g_2 and c are valid alternatives g_n to perform expansion. The rest of nodes with a mandatory assignment are directly proven not to be valid, as they generate LDC in the extended functions in the non sensitive cofactor. The new introduced don´t care is situated at term T2, where the discrepancy is located.

The necessary condition can only be met if all MAs can be computed. This is generally not possible, since computation of MAs is an NP–complete problem. However, this theorem demonstrates that all possible transformations can be found without backtracking given enough time to compute all MAs.

V. ENHANCED MULTIPLE-VARIABLE EXPANSION

If the single variable expansion procedure described in the preceding section did not find alternative nodes, then this procedure is unable to generate LDC at necessary discrepant pairs. Following the reasoning scheme shown in the previous section, the dominator is expanded with more variables (N_i) in order to generate LDC where needed. Due to the exponential increase in the number of candidates (gn_i) when performing multiple wire addition, the traditional search of possible alternatives by trial and error may become impractical for this case. The following Theorem provides the means to identify transformations involving multiple variable expansion.

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.

Demonstration. The demonstration is slightly different depending of whether the initial assignment gn1 = ‘v’ is in f1 or in f2. If it is in f1, then we know that of all PI combinations, only those with gn1 = ‘v’ and gn2 = ‘t’ are able to test fault f1. These two assignments define the sensitive cofactor in this case. From the premises of the theorem, we know that gn1 = ‘v’ and gn2 = ‘t’ = NOT ‘s’, along with the other observability conditions given by f2, is a LDC. Therefore the discrepant term is a LDC and the transformation has been identified. If the initial assignment gn_1 = ‘v’ is in f2, then we know that all discrepant terms except the one in the cofactor with N1 = ‘v’ and N2 = ‘s’ are LDCs. However, this cofactor is not sensitive, as N2 = ‘t’ = NOT ‘s’ is required for the fault f1 to be observable when gn1 = ‘v’.

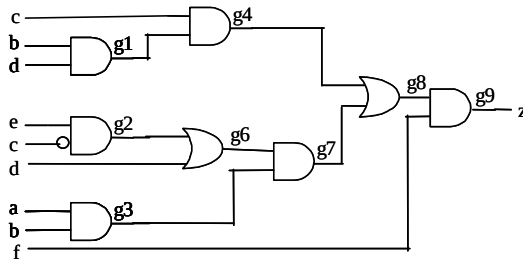

Fig 2 Example for multiple wire expansion

The following lemma complements the previous theorem, by discarding alternatives for expansion

Lemma 5.- *“If the mandatory assignment g_n = ‘t’ = ’s’ is obtained for both faults f1 and f2, then g_n will not generate alternatives for multiple variable expansion.”*

Demonstration.- From the premises of the lemma, it is deduced that LDC will not be generated at the sensitive cofactor. This condition will remain valid even though other additional variables are considered for expansion. In the following example it will be shown how this theorem allows to carry out expansion of two variables efficiently.

Example 3.- In the circuit in fig 2, f1 d-> g_6 s-a-1 fault is selected as the target fault. g_9 is selected as the dominator. After an implication of f1 we see that g_9 value at non faulty circuit is ‘0’. Because of that, selected value for ‘a’ is ‘0’. T3 is the proposed discrepant term in this case and we select f2 to be g_8,g_9 s-a-0. Implications are shown in table 4.

In this case there are no alternatives for single variable expansion. 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 g_1. We are not able to determine if g_1 value by itself will generate an adequate LDC, as its implied value is ‘U’. According to Theorem 2 we set g_1=’0’ (equal value than the f1-MA) and extend f2-SMA with this value. After implication, we find that the inexistent MA g_2=’U’ is converted in g_2=’1’ for the extended set. In this case we infer that under the initial observability conditions it is impossible to obtain combination g_1=’0’ and g_2=’0’. Because of that, new LDC have been successfully generated at the sensitive cofactor when this double variable expansion is performed.. Reduction operation yields to the elimination of nodes g_6 and g_7. With the selected value for ‘a’, g_9 is expanded to $g_9 = g_8 f (g_1+g_2)$.

Theorem 2 can be applied recursively to find all possible multiple variable expansions for a dominator node. Thus, the general procedure for identification of optimization transformations is as follows. First, a target wire and a dominator node are selected. If the target wire test is redundant, then the target wire can be removed without performing any further computation. Otherwise, the fault f2 corresponding to the fault propagating input at the dominator is tested. If the conditions of Theorem 2 are met, then a single

variable expansion transformation has been found. Otherwise, we impose a new mandatory assignment in f1 or f2 according to the premises of Theorem 2. If no double variable expansion is found, then we impose another new mandatory assignment in order to try a triple variable expansion and so on. Note that the recursion will eventually end when all nodes have equal mandatory assignments for f1 and f2, if no difference is found before. In practice, the recursion is pruned as soon as the expansion operation has an area cost higher than the target reduction.

VI. EXPERIMENTAL RESULTS.

In this section, we provide experimental results for benchmark combinational circuits. We modified *RAMBO* tool [1] to work with the enhanced algorithm. The experiment was performed in a Sun Ultra 1 WS with 64 Mb RAM. Original benchmarks were first optimized with *script.rugged* included in *SIS* [10] After this optimization, both the enhanced and the original RAMBO 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 35% less of time in average. Table 6 summarizes the results of the comparison between the old and enhanced algorithms regarding efficiency issues. In this table, columns two and three show the CPU time used to perform optimization with the old (*OLD*) and the enhanced algorithm (*ENH*). Columns four and five show the number of total test performed in both cases. Columns six and seven 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 is an 87% 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 RAMBO algorithm.

VII. CONCLUSIONS AND FUTURE WORK

We have presented a generalization of the structural logic optimization methods based on functional considerations. In addition, the necessary and sufficient conditions to efficiently identify all possible transformations, including multiple variable expansion, have been obtained. This approach also provides substantial efficiency improvements compared with previous works. The proposed functional formulation provides the theoretical framework for a new range of future applications, particularly including post-mapping and post-layout logic optimization.

REFERENCES

[1] K.-T. Cheng, L. Entrena. “Sequential Logic Optimization by Redundancy Addition and Removal”. Proc. ICCAD, p. 310-315. November, 1993

[2] L. A. Entrena, K.-T. Cheng. "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal". IEEE Transactions on CAD, vol.14, n. 7, p. 909-916. 1995.

[3] 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, p. 587-596. June, 1997.

[4] S. C. Chang, M. Marek-Sadowska, K.-T. Cheng. “Perturb and Simplify: Multilevel Boolean Network Optimizer”. IEEE Transactions on CAD, vol. 15, nº 12, p. 1494-1504. November 1996.

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

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

[7] B. Rohfleisch, F. Brglez. “Introduction of permissible bridges with application to logic optimization after technology mapping”. Proc. European Design & Test Conference (ED&TC), p. 87-93. February 1994.

[8] B. Rohfleisch, B. Wurth, K. Antreich. "Logic Clause Analysis for Delay Optimization". Proc. 32nd DAC, p. 668-672. June 1995.

[9] S. C. Chang, L. P. van Ginnekan, M. Marek-Sadowska. “Fast Boolean Optimization by Rewiring”. Proc. ICCAD, p. 262-269. November, 1996.

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

TABLE 6 EXPERIMENTAL RESULTS

	CPU (s)		TOTAL TESTS		ALT. NODES	
	OLD	ENH	OLD	ENH	OLD	ENH
9symml	2	2	4685	681	19761	21
Alu2	14	5	13953	1305	47482	177
Alu4	110	46	54635	2385	173261	382
apex6	27	24	38224	2855	82606	175
apex7	3	2	8948	882	13491	116
C1355	12	12	13676	1972	26280	16
C17	0	0	60	36	42	1
C1908	9	8	10423	1813	24072	31
C3540	369	202	147181	18070	534355	907
C432	1	1	2310	847	4255	71
C499	11	11	13676	1972	26280	16
C5315	58	51	44848	6251	93891	434
C7552	310	236	128289	17212	294463	1358
c8	1	1	3751	533	8489	60
C880	4	3	7171	1490	15737	44
cht	1	1	4023	621	6284	11
Cm138a	0	0	149	53	231	10
count	2	1	5208	488	12507	136
Cu	0	0	529	210	1670	23
decod	0	0	581	104	1101	7
example2	11	6	16037	1239	48678	289
frg1	2	1	4471	666	18007	62
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
i9	96	61	80484	2054	223443	144
k2	321	182	151545	2821	525216	628
lal	0	0	1250	328	2615	19
pair	58	46	52889	5844	114494	486
t481	128	97	77414	39599	203011	201
term1	1	1	2800	696	8196	52
vda	41	26	32858	1287	104936	271
x3	31	25	39251	3643	84612	199
x4	9	7	16656	1323	30837	91
TOTAL	*685*	*445*	*455147*	*57595*	*1297360*	*2091*