Theorems and Extensions of Single Wire Replacement

Shih-Chieh Chang, Zhong-Zhen Wu
Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan 621, Republic of China

Abstract

In this paper we discuss the theorems and extensions of single alternative wire that attempts to replace one wire by another wire without changing the logic functionality. The wire replacement technique has been successfully applied to achieve logic optimization and routability improvement. However there still exist several fundamental problems that have not been addressed such as whether the algorithm can find all single alternative wires. First, we present some cases of alternative wires which the previous work [5] cannot obtain. Then, several theorems of tight necessary conditions and dominating conditions for a wire to be an alternative wire are proposed. With these theorems, we are able to derive an efficient procedure to find all possible alternative wires. The experimental results are very encouraging.

1 Introduction

Many researchers in the past have proposed logic synthesis techniques based on the reasoning of Automatic Test Pattern Generation (ATPG) because of the quick run time and low memory usage [2][3][4][5][6][7][8][9][10][11][12][13][14][17][18][20][21]. Though most ATPG based algorithms are different in ways of performing optimization, their basic philosophy is the same, i.e., it adds and removes redundant connections in a circuit. Among those, the work [6] first introduces the concept of the redundancy addition and removal using ATPG for area minimization and later extends to sequential logic optimization [10]. In [12], they present several new transformations for synchronous sequential logic, such as adding redundancies across time frame and adding redundancies to block initialization of other wires. The research in [5] proposes a post-layout logic synthesis technique, called the alternative wire technique (**Alwire**). An *alternative* wire of a target wire is a redundant wire whose addition can make the target wire redundant and therefore can be removed. The basic idea of the Alwire technique first has some target wire for removal in mind. Then, the technique searches in the circuit another redundant wire which when added, can cause the target wire redundant and then can be removed. For example

consider removing wire c->g_2 in Fig. 1(a). The Alwire technique will identify that by adding wire g_1->g_5, wire c->g_2 can be removed without changing the logic functionality. In other words, the two circuits in Fig. 1(a) and Fig. 1(b) are functionally equivalent but their wire topologies are different.

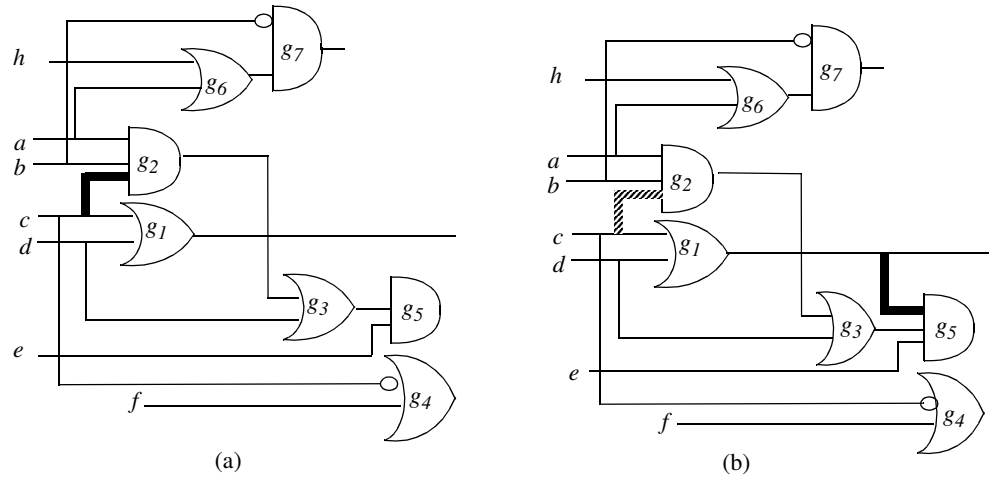

Fig. 1 An example for single alternative wire.

Though an algorithm is proposed to find a single alternative wire of a target wire in [5], there still exist several fundamental problems which have not been addressed such as whether the algorithm can find all single alternative wires. In this paper, we discuss several theorems of tight necessary conditions and dominating conditions for a wire to be an alternative wire. The main purpose of these theorems is to speedup the process of finding alternative wires and search for more alternative wires which the previous work [5] cannot obtain. With these theorems, we are able to derive an efficient procedure to find all possible alternative wires. We would also like to mention that this paper concentrates on the additional rules for pruning the set of candidate connections and the removal of the dominators on the set of destination nodes.

The paper is organized as follows. Section 2 discusses some backgrounds and the review of the Alwire technique. Section 3 shows two definitions of mandatory assignments. In section 4 and 5, we discuss the necessary and dominating conditions. Finally, we give the experimental results followed by conclusions.

2 The review and the drawbacks of the Alwire Technique.

In this section, we briefly review the Alwire technique. Before we review the Alwire technique, we would like to have some definitions and terminology. Let wire *w* be a wire tested for a stuck-at 0 {1} fault; a *faulty* circuit is the circuit in which *w* is replaced by a constant 0 {1}. An input combination *v* is a *test vector* if an output of the good circuit and faulty circuit are different when applying *v*. If no such a test vector exists, then the wire under stuck-at fault test is *undetectable* and the wire is claimed to be *redundant*; otherwise the wire is *irredundant*. Also, a node is called a *dominator* of wire *w* if all paths from wire *w* to any primary output (PO) must pass through the node. Side inputs of a dominator of wire *w* are those not in the fanout paths of *w*. During a wire stuck-at fault test, the side inputs of the wire's dominators must be assigned non-controlling values so that the fault can propagate through.

The basic procedure of the Alwire technique is as follows: It first decides some existing irredundant wire that is the target to be removed. Then, the technique searches for some non-existing wire, sometimes called a candidate connection, that once added can make the target wire stuck-at fault undetectable (redundant) and therefore it can be removed. Finally, the technique checks whether the candidate connection is redundant, i.e., whether adding the non-existing wire preserves the circuit's functionality. Only when the candidate connection is verified as redundant, the target wire can be replaced by the candidate connection. For example, consider finding an alternative wire for wire c->g_2 in Fig. 1(a). Let us first check wire c->g_2 s-a-1 test. To propagate the fault effect and to activate the fault, the value assignments {a=1, b=1, c=0, d=0, e=1, g_1=0, g_4=1, g_6=1, g_7=0} must be set. These assignments are called the *mandatory assignments (MAs)* for wire c->g_2 s-a-1 test. In the circuit, node g_5 is a dominator for the target wire. If the (non-existing) wire g_1->g_5 is added to the circuit, the MA of g_1=0 will cause a dominator g_5 to have the MA of 0 which blocks the fault propagation and causes the fault undetectable. So, wire g_1->g_5 is identified as a candidate connection. Finally, the technique checks whether g_1->g_5 s-a-1 is redundant. Since it is redundant, wire g_1->g_5 is an alternative wire for c->g_2. We would like to mention that a candidate connection can also be a wire with an inverter. For example, node g_1 has the MA of 0 and node g_3, an OR gate, is a dominator. We can conclude wire g_1->Inverter->g_3 is a candidate connection because the addition causes node g_3 to have the MA of 1.

In the original Alwire procedure, a candidate connection w_c=n_s->n_d is constructed in the way that the destination node n_d of a candidate connection is a dominator and the source node, n_s has an MA. Therefore, when adding a candidate connection, the MA in the source node n_s can cause the dominator n_d to have an MA so as to block the fault propagation. If there are k MAs and m dominators, the Alwire procedure will construct $k*m$ candidate connections, each of which will be tested for redundancy. For example, to replace wire c->g_2 in Fig. 1(a), there are 9 MAs and 3 dominators for wire c->g_2 so there will be 27 candidate connections. For example, wires {a->g_3, c->g_5, d->inverter->g_3, g_1->g_2, g_1->g_5} are candidate connections. The algorithm of Alwire is shown in Fig. 2.

```
Alwire (wire) {
  Perform wire s-a-fault test;
   for each node i has MA {  /* node i will be a source node */

      for each node j is a dominator { /* node j will be a destination node */
          construct a candidate connection k= i->j;
          if  (wire k is redundant) wire k is an alternative;
      }
    }
}
```

Fig. 2 The algorithm of Alwire.

The above procedure, however, does not find all alternative wires. For example, consider wire g_1->g_5 in Fig. 1(b) and assume that the dotted wire c->g_2 does not exist. Since both circuits (Fig. 1a and b) are functionally equivalent, wire g_1->g_5 in Fig. 1(b) can be replaced by wire c->g_2. Such a replacement cannot be identified by the above procedure because g_2 is not a dominator for g_5. This alternative wire cannot be found by the Alwire routine in Fig. 2. Therefore, the previous algorithm cannot find all single alternative wires. In addition, in the Alwire routine, each candidate connection requires a redundancy check. We will show that many candidate connections can be easily found to be irredundant without the time-consuming redundancy checks.

3 Forced Mandatory Assignments and Observability Mandatory Assignments

After the stuck-at fault test for the target wire, a node in the circuit is assumed to have either the values of {X, 1, 0, d, D}. The value X represents an unknown value, the MA, d (=0/1) represents good circuit is 0 and faulty circuit is 1, and the MA of D (=1/0) represents good circuit is 1 and faulty circuit is 0 when evaluated by test vectors. Some MAs

can be obtained by direct implication [1] [15]. More MAs can be found by more complicated approaches [16] [19]. In this section we review two important attributes of an MA derived from a stuck-at fault test. For details, please refer to the paper in [7].

Definition 1: Suppose that after a stuck-at fault test on wire *w*, node *n* has an MA in a circuit *C*. We build a new circuit *C'(n)* as follows. (See Fig. 3.) If the MA of *n* is 1 or D, we disconnect *n* from its fanouts and assign these fanouts a constant 0. If the MA is 0 or d, we assign those fanouts a constant 1. We say that the node *n* has a ***forced*** MA in *C* if when we perform the same stuck-at fault test on the wire *w* in *C'(n)*, the fault becomes untestable.

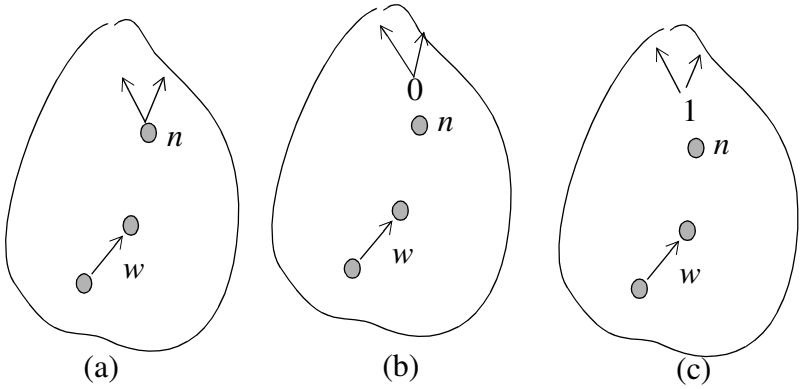

Fig. 3 (a). *w* is the wire on which the stuck-at fault test is performed. (b) shows *C'(n)* when *n* has a MA=1 or D. (c) shows *C'(n)* when the node has a MA=0 or d.

In [7], it proves "the MAs obtained by setting the side inputs of the dominators to their non-controlling values, the outputs of the dominators to d or D, and the *activating* MA on the source node of the target wire stuck-at-fault, are all forced. In addition, all the MAs in the **backward** propagation step are also forced." In contrast, all forward implied MAs are not forced.

Definition 2: The ***observability*** *MAs* of wire *w, obvMAs(w)* are MAs assigned only for propagating a fault from wire *w* to a Primary Output (PO).

The process of finding the MAs of a wire *n->m* s-a-1 test can be divided into two steps. First, we set up the observability MAs for wire *n->m*, *obvMAs(n->m)*; then we assign the activating value of 0 at node *n* and derive more MAs from MA *n=0*. We define the "additional" MAs derived by MA *n=0 (*including *n=0)* to be the *activating derived* MAs, *actDrvMAs(n=0)*. Therefore, the MAs of *n->m* s-a-1 test will be equal to *obvMAs(n->m)*$\cup$*actDrvMAs(n=0)*.

4 Pruning the candidate connections by necessary conditions.

A candidate connection in the Alwire algorithm can be just a wire or other transformations such as adding an inverter or a two-input AND/OR gate [5]. During the construction of the candidate connection, the original Alwire algorithm allows us to add AND/OR gates so that each pair of source and destination nodes can form a candidate connection. For simplicity of the following discussion, let us consider a candidate connection to be only a wire. Other transformations can be extended. To improve the results of the original Alwire routine, on one hand, we need to extend the selection criteria for candidate connections to ensure all possible alternative wires can be found. On the other hand, for efficiency, we need to reduce the number of candidate connections which later require time-consuming redundancy checks. To achieve both goals, in the following, we discuss some "tight" necessary conditions of an alternative wire. These necessary conditions will be used to derive the candidate connections. Let the circuit under consideration be C and after adding a redundant wire w_r, the new circuit be $C+w_r$. In the following, we show that the source node of an alternative wire must satisfy the condition in Theorem 1 and the destination node must satisfy the condition of Theorem 2. The theorems in this section has appeared in [6].

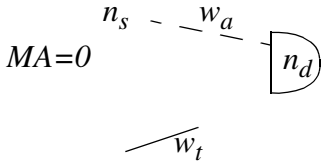

Fig. 4 wire w_a is an alternative wire for wire w_t.

Theorem 1: Let n_d be an AND {OR} gate. If w_a=n_s-> n_d is an alternative wire for w_t then the source node n_s must have a mandatory assignment 0 {1} for the stuck-at fault test of w_t. (Fig. 4)

Proof. If the source node of w_a, n_s does not have an MA, some test vectors for w_t stuck-at fault will evaluate a 0 and other test vectors evaluate a 1 at node n_s. If n_d is an AND {OR} gate, after adding w_a, a test vector that generates 1 {0} at node n_s is still a test vector for w_t. This implies that w_t is not redundant and therefore w_a is not an alternative wire for w_t. Using the same argument, we can also show that n_d must be an OR gate when n_s=1 and AND gate when n_s=0. **Q.E.D.**

Theorem 2: If w_a= n_s->n_d is an alternative wire for w_t, then an AND {OR} gate n_d (the destination node) must have a forced mandatory assignment 1 or d {0 or D} for the stuck-at fault test on w_t in C.

Proof. Let us prove the case when node n_d is an AND gate. We will show that if node n_d is in the fanout cone of the target wire w_t, node n_d must have a forced MA of d and if node n_d is not in the fanout cone of w_t, node n_d must have a forced MA of 1. Consider the case when n_d is not in the fanout cone of w_t. Since w_a is an alternative wire of w_t, adding w_a will cause w_t redundant. In implication, the effect of adding a wire w_a (=n_s->n_d) is equivalent to inserting the MA of n_s to n_d. In other words, w_t stuck-at fault test becomes redundant, if node n_d is "inserted" the MA of node n_s. According to Theorem 1, n_s must have an MA of 0, so node n_d must have a forced (backward) MA of 1 by the definition of a forced MA.

If node n_d is in the fanout cone of w_t, adding w_a (=n_s->n_d) must block the fault propagation for w_t stuck-at fault test. Note that after replacing w_t by w_a, the circuit functionality is unchanged. Therefore, the effect of adding w_a must be able to correct the "errors" caused by removing w_t. It was proved in [3] that in the case of n_d is an AND gate, node n_d must have an MA of d (=0/1) where the MA d stands for the case that the good circuit is 0 and the faulty circuit is 1. It is because that n_s has an MA of 0 and adding n_s->n_d will cause n_d to be 0, which not only blocks the fault propagating, but also causes the result to be same as the good circuit. **Q.E.D**.

For example in Fig. 1(b) consider to find an alternative wire for wire g_1->g_5. After computing g_1->g_5 s-a-1 test, we have MAs={$a=1$, $b=1$, $c=0$, $d=0$, $e=1$, $g_1=0$, $g_2=1$, $g_3=1$, $g_4=1$, $g_6=1$, $g_7=0$}. Since node g_2 has a forced (backward) MA of 1, we can use this to corrupt the existing MAs. The wire c->g_2 brings the value of 0 to the input of an AND gate changing the original forced MA of 1. It makes the wire c->g_2 a candidate connection to remove wire g_1->g_5. Since wire c->g_2 is redundant, c->g_2 is an alternative wire for g_1->g_5.

Theorem 1 gives a necessary condition for the source node of an alternative wire while Theorem 2 gives a necessary condition for the destination node. With these two theorems, we can enumerate all possible cases which satisfy the necessary conditions of an alternative wire in the following: Construct a candidate connection whose source node has an MA and whose destination AND {OR} node has a forced MA of 1 or d {0, or D}. The set of candidate connections obtained from the above procedure contains all possible alternative wires.

Although technically, Theorem 1 and Theorem 2 are similar to the rules described in the previous work [5] [7], the differences between the theorems in this paper and the rules in the previous work are in fact quite profound. Consider Theorem 1. Previously it was writ-

ten as a rule which says that the source node of an alternative has an MA. However, one may wonder whether there exists an alternative whose source node does NOT have an MA. Theorem 1 proves that every alternative's source must have an MA. Theorem 2 has even profound implications compared to the rule in the previous work. Previously, the rule uses a dominator to be the destination node of an alternative. By the rule, one may wonder whether using only dominators is enough to find all alternatives. Theorem 2 shows that it is not enough and in addition, it shows that only if another type of node is considered, we can find all alternatives. With these two theorems of necessary conditions, we are able to find all alternatives. Without them, it was not possible before.

In the following, we show a way to improve the efficiency, i.e. to tighten the necessary conditions.

Theorem 3: If n_s has an observability MA for w_t stuck-at fault test, w_a=n_s -> n_d cannot be an alternative wire for w_t.

Proof. An observability MA has no relationship with the activating value. Therefore, node n_s must have the same value of MAs for both the w_t s-a-1 and s-a-0 faults. Since w_a is an alternative wire for w_t, w_a must be a redundant wire in C. Therefore, adding w_a can cause both the w_t s-a-1 fault and s-a-0 fault redundant. It is not possible to add a redundant wire to cause both the target wire's s-a-1 and s-a-0 redundant. Therefore, n_s must not have an observability MA. **Q.E.D.**

For example in Fig. 1(a), to find a single alternative wire for wire c->g_2, the MAs for c->g_2 stuck-at-1 test are {a=1, b=1, c=0, d=0, e=1, g_1=0, g_4=1, g_6=1, g_7=0}. Among the MAs, {a=1, b=1, d=0, e=1, g_4=1, g_6=1} are the observability MAs. Therefore, to find a single alternative wire that fanins to g_5, we only need to consider the candidate connections of {c->g_5, g_1->g_5, g_7->g_5}.

```
Alwire_n1 (wire) {
  Perform wire s-a-fault test;
   for each node i which has an MA but not an observability MA {

     for each AND (OR) node j which has a forced MA of 0 or d (1 or D) {
          construct a candidate connection k= i->j;
          if  (wire k is redundant) wire k is an alternative;
       }
     }
}
```

Fig. 5 An algorithm of Alwire which finds all alternative wires.

With Theorem 1, Theorem 2 and Theorem 3, we can modify the algorithm in Fig. 2 to obtain an efficient Alwire algorithm which can guarantee to find all possible alternative wires. The new algorithm is shown in Fig. 5. Because both the problems of finding MAs and redundancy removal are NP complete problems, the completeness of our algorithm depends on the efficiency of an implication engine which may not find all MAs and may not identify all redundancies.

We would like to mention that an alternative wire (n_s->n_d) consists of a source node (n_s) and a destination node (n_d). Theorem 1 and Theorem 3 give the necessary conditions for the source node of an alternative wire and Theorem 2 gives a necessary condition for the destination node. As a result, if a wire (n_s->n_d) is an alternative, the source node n_s must satisfy the conditions of Theorem 1 and Theorem 3, and the destination node n_d must satisfy the condition of Theorem 2. Since our algorithm in Fig. 5 constructs the set of all (candidate) wires which satisfy Theorem 1, Theorem 2 and Theorem 3, all alternative wires must be in the (candidate wire) set. Therefore, if we check all the candidate connections in the set, we are able to find all alternative wires. On the other hand, the conditions of other work proposed are sufficient conditions and cannot claim to find all alternative wires.

5 Pruning the candidate connections by dominating conditions.

In this section, we discuss a way to reduce the number of redundancy checks of candidate connections. We achieve the objective by obtaining the irredundancy information of other candidate connections after checking redundancy of some candidate connection. Consider to find an alternative wire for c->g_2 in Fig. 1(a). Both candidates g_1->g_2 and g_1->g_5 are candidate connections and require redundancy checks. Since node g_5 is a dominator of node g_2, conceptually, node g_2 is less observable than node g_5 because g_5 is closer to POs than g_2. The following theorem shows that if g_1->g_2 is irredundant, g_1->g_5 must be irredundant. As a result, if wire g_1->g_2 is found to be irredundant, we don't have to perform redundancy check on wire g_1->g_5 since it must be irredundant.

Theorem 4: In Fig. 6, let w_1=n_s->n_1 and w_2= n_s->n_2 be both candidate connections for w_t and also there is only one path from node n_1 to node n_2. If wire w_1 is irredundant, w_2 must be irredundant.

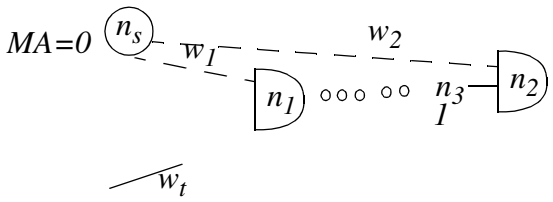

Fig. 6 Dominating conditions.

Proof. In Fig. 6, we will show that a test vector *v* for w_1 s-a-1 fault must also be a test vector for w_2 s-a-1 fault. First, according to Theorem 1 and Theorem 2, if node n_1 is an AND gate, node n_2 must be an AND gate. Also, because AND gates must have the same forced MAs of *d* for the target fault from Theorem 2, there are even number of inversions in the path of node n_1 to n_2. Since both faults w_1 s-a-1 and w_2 s-a-1 have the same activating value, we only need to show that test vector *v* of w_1 s-a-1 fault can let w_2 s-a-1 be observable. Let node n_3 be the node in the transitive fanout of n_1 and is direct fanin of n_2 in Fig. 6. In the circuit without wire w_1, we will show that evaluating *v* (a test vector of w_1 s-a-1 fault), node n_3 must have a value *1*; otherwise vector *v* is not a test vector for w_2 s-a-1 fault. Now, let us assume that w_1 is present in the circuit. When test w_1 s-a-1 fault, we have MA n_1=d (0/1), i.e., the good circuit is 0 and faulty circuit is 1. Since there are even number of inversions in the path, we must have MA n_3=d. Therefore, if w_1 is removed in the circuit, node n_3 must have the (faulty) value *1* when evaluating a test vector *v*. In other words, node n_3 has a value 1 when evaluating the circuit without w_1.**Q.E.D.**

6 A Summary Example

We now give a summary example for the necessary and dominating conditions. Let us consider to find alternative wires for wire $c\text{->}g_2$ in Fig. 1(a). First, to perform wire $c\text{->}g_2$ s-a-1 fault test, the MAs {*a=1, b=1, c=0, d=0, e=1,* g_1*=0,* g_4*=1,* g_6*=1,* g_7*=0*} must be set. There are 3 dominators {g_2, g_3, g_5} and 9 MAs. In the previous Alwire algorithm [5], it constructs 9*3=27 candidate connections. With necessary conditions, the new Alwire algorithm in Fig. 5 will find that among the MAs, the MAs {*a=1, b=1, d=0, e=1,* g_4*=1,* g_6*=1*} are the observability MAs, and {*a=1, b=1, c=0, d=0, e=1,* g_2=d(0/1), g_3=d, g_5=d} are the forced MAs. The new Alwire algorithm only constructs the candidate connections {$c\text{->}g_5$, $g_1\text{->}g_2$, $g_1\text{->}g_5$, $g_7\text{->}g_2$, $g_7\text{->}g_5$}, and therefore the number of candidate connections is reduced to 5 from 27. Still, we require 5 redundancy checks on these candidate connections. However, according to the dominating conditions, since the candidate connection $g_7\text{->}g_2$ is found to be irredundant, the candidate connection $g_7\text{->}g_5$ must be irre-

dundant without the redundancy check. As a result, we only require 4 redundancy checks to find all possible alternative wires. With these 'tight' necessary and dominating conditions, the speedup of the Alwire technique can be achieved.

7 Experimental Results

We have implemented the new Alwire technique and performed experiments on a set of MCNC and ISCAS benchmarks. The circuits are first optimized by SIS script and then decomposed into AND/OR gates. In Table 1, we first demonstrate that our new Alwire algorithm in Fig. 5 can obtain more alternative wires and require less redundancy checks than the original Alwire technique [5]. Column one shows the name of a circuit. Column two and three show the number of alternative wires obtained in the previous Alwire algorithm [5] and in the new algorithm of Fig. 5. Note that even though "ideally" the new Alwire algorithm can find all possible alternative wires, we may still lose some optimality because of the limitation of the implication engine used in our implementation. In other words, if our implication engine can find all MAs, the number in the third column shows the number of all possible alternative wires. For example, circuit 9symml has 615 alternative wires in the previous algorithm and has 1083 alternative wires in the new algorithm. We then show the speed up of our new algorithm by applying the necessary and dominating conditions. During the Alwire algorithm, each candidate connection requires a redundancy check. Column four shows the number of candidate connections generated from [5]. Column five gives the number of candidate connections generated after applying the necessary conditions, while column six gives the number after applying both the necessary and the dominating conditions. The run time is given in the last column. For example, circuit 9symml in the original algorithm generates 296770 candidates. In the new Alwire algorithm, with only the necessary conditions, it generates 132858 candidates and 104730 candidates with both necessary and dominating conditions. All calculations were performed on an Ultra 2 with 512 MB memory.

8 Conclusions

In this paper, we have presented an efficient algorithm of finding all possible alternative wires. The algorithm is based on the necessary and dominating theorems of an alternative wire. The theorems provide more flexibility and efficiency for single alternative wire replacement in the logic domain, and will be expected to extend to other ATPG based techniques.

TABLE 1 The results of Alwire with necessary and dominating conditions.

circuit	#aw (ori)	#aw (new)	#cand. (ori)	# cand. (nec.)	#cand. (nec.+dom.)	time (sec) ours
9symml	615	1083	296770	132858	104730	10.25
C1908	107	122	93736	150839	150738	22.45
C3540	1975	2603	1963880	1630919	1484177	157.44
C5315	1580	1651	384450	196885	190995	29.29
alu2	2381	2591	658084	459326	419061	56.04
alu4	6807	7547	1826944	1546193	1450491	202.28
c8	102	124	22444	14447	14409	1.37
comp	198	240	40052	64161	60400	3.91
count	122	138	54248	50797	50328	7.82
cu	87	94	11716	3793	3109	0.26
dalu	6326	6548	1980724	1577393	1528635	200.35
example2	930	935	101884	88267	87663	8.05
f51m	681	699	64008	28140	24979	2.64
frg2	2845	3093	613084	756740	749569	129.83
i10	2567	3150	1346450	2059646	2049864	322.93
i8	1658	1701	1339212	1232861	1230029	162.31
i9	225	233	406908	474793	474278	55.74
lal	41	41	9298	5931	5743	0.53
pair	2279	2441	453196	281772	274455	41.38
rot	803	871	120266	62388	59028	7.77
sct	74	75	7532	6008	5986	0.56
term1	847	1071	101268	47359	38162	3.74
too_large	1123	1498	453690	290591	239276	20.96
ttt2	578	672	50218	25226	23979	2.51
x1	307	356	79044	42216	40549	3.52
x2	82	82	5082	2691	2673	0.23
x3	679	882	186476	115882	112746	12.36
x4	256	258	61206	38600	38331	4.45
Average	1	**1.12**	1		**0.85**	

9 References

[1] M. Abramovici, M.A. Breuer, A.D. Friedman, “Digital Systems Testing and Testable Design,” *Computer Science Press, 1990.*

[2] C. L. Berman and L. H. Trevillyan. “Global Flow Optimization in Automatic Logic Design,” *IEEE Transaction on Computer Aided Design, Vol. 10, pp. 557-564, May 1991.*

[3] D. I. Cheng, Chih-Chang Lin, M. Marek-Sadowska, “Circuit Partitioning with logic Perturbation,“ *Proc. ICCAD, pp. 650-655, 1995.*

[4] S.C. Chang, M. Marek-Sadowska, and K.T. Cheng, “Perturb and Simplify: Multi-level Boolean Network Optimizer,” *IEEE Transaction on Computer Aided Design, Vol. 15, pp. 1494-1504, Nov. 1996.*

[5] S.C. Chang, K.T. Cheng, N.S. Woo, M. Marek-Sadowska, “Post Layout Logic Restructuring Using Alternative Wires,“ *IEEE Transaction on Computer Aided Design, Vol. 16, pp.587-596, June 1997.*

[6] S.C. Chang, L. Van Ginneken, and M. Marek-Sadowska, “Fast Boolean Optimization by Rewiring,” *Proc. ICCAD, pp.262-269, 1996.*

[7] S.C. Chang, L.V.Ginneken, and M. Marek-Sadowska, “Circuit Optimizations by Rewiring,” *IEEE Transaction on Computer, Vol. 48, pp. 962-970, 1999.*

[8] K. T. Cheng and L. A. Entrena, “Multi-Level Logic Optimization by Redundancy Addition and Removal,” in *Proc. European Conference On Design Automation*, pp. 373-377, Feb. 1993.

[9] M. Damiani, J. C. Y. Yang and G. De Micheli, “Optimization of Combinational Logic Circuits Based on Compatible Gates,” *Proc. Design Automation Conf., pp. 631-636, June 1993.*

[10] L. A. Entrena and K. T. Cheng, “Sequential Logic Optimization by Redundancy Addition and Removal,” *Proc. ICCAD, pp. 310-315, Nov. 1993.*

[11] L. A. Entrena and K. T. Cheng, “Combinational and Sequential Logic Optimization by Redundancy Addition and Removal,” *IEEE Transaction on Computer Aided Design, Vol. 16, pp.587-596, July 1995.*

[12] U. Glaser and K. T. Cheng, “Logic Optimization by an Improved Sequential Redundancy Addition and Removal,” *Proc. of ASP-DAC, pp.235-240, Sept. 1995.*

[13] M. Higashida, J. Ishikawa, M. Hiramine, K. Nomura, “Multi-level Logic Optimization Based on Pseudo Maximum Sets of Permissible Functions,” *European Design Automation Conf., pp. 386-391, 1993.*

[14] Y-M Jiang, A. Krstic, K-T Cheng, M. Marek-Sadowska, “Post-Layout Logic Restructuring for Performance Optimization,” *Proc. DAC, pp. 662-665, 1998.*

[15] T. Kirkland and M. R. Mercer, “A Topological Search Algorithm For ATPG,” *Proc. Design Automation Conf., pp. 502-508, June 1987.*

[16] W. Kunz and D. K. Pradhan, “Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation for Digital Circuits,” *Proc. Int. Test Conf., pp.816-825, Oct. 1992.*

[17] S. Muroga et al, “The Transduction Method-Design of Logic Networks Based on Permissible Functions,” *IEEE Transaction. on Computer C38(10). pp. 1404-1423 Oct. 1989.*

[18] B. Rohfleisch, B. Wurth, K. Antreich, “Logic Clause Analysis for Delay Optimization,” *Proc. DAC, pp. 668-672, 1995.*

[19] M. Schulz and E. Auth, “Advanced Automatic Test Pattern Generation and Redundancy Identification Techniques,” *Proc. Fault Tolerant Computing Symp.*, pp. 30-34, June 1988.

[20] G. Strenz, B. M. Riess, B. Rohfleisch, and F. M. Johannes, “Timing Driven Placement in Interaction with netlist Transformations,” *Proc. ISPD, 1997.*

[21] M. Yuguchi, Y. Nakamura, K. Wakabayashi, T. Fujita “Multi-Level Minimization based on Multi-Signal Implications,” *Proc. DAC, 1995, pp. 658-662.*