Fig. 1 Example of alterative wires

Fig. 2 Applications of alternative wires

Fig. 3 Example of redundant wire.

Fig. 4 The addition of a wire may cause a target wire stuck-at fault untestable

Fig. 5 The procedure of finding alternative wires

Fig. 6 Types of transformations

Fig. 7 Pseudo code for finding single-wire alternative

Fig. 8 Identifying multiple-wire alternatives

Fig. 9 An example of global flow

Fig. 10 Cost of alternative wires

Fig. 11 Example of alternative function

Fig. 12 An example of finding alternative functions

Fig. 13 An algorithm to find alternative functions of a target wire

Fig. 14 The flow of layout driven synthesis

TABLE 1 Experimental results

[8] T. Kirkland and M.R. Mercer, “A Topological Search Algorithm For ATPG,” *Proc. 24th Design Automation Conf.*, pp. 502-508, June 1987.

[9] W. Kunz and D. K. Pradhan, “Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation Digital Circuits”, in *Proc. Int’l Test Conference*, pp. 816-825, October 1992.

[10] R.Murgai, N. Shenoy, R.K.Brayton, and A.Sangiovanni Vincentelli, “Improved Logic Synthesis Algorithms for Table Look Up Architectures, ” *Proc. International Conference on Computer-Aided Design*, pp. 564-576, November 1991.

[11] S.Muroga, Y. Kambayshi, H.C. Lai, J.N. Culliney, “The Transduction Method-Design of Logic Networks Based on Permissible Functions,” IEEE TC, pp. 1404-1424, Oct. 1989.

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

[13] N-S Woo, “A Heuristic Method for FPGA Technology Mapping based on the Edge Visibility, Proc. of 28th Design Automation Conference, 1991, pp. 248-251

rd73	63	3(4)	20(267)	8.7	386.5
sao2	220	57(157)	154(1802)	184.7	2405.8
x2	58	19(39)	38(193)	4.9	26.7
example2	376	60(221)	106(868)	345.1	2892.8
C1908	422	48(125)	141(1692)	3723	38273
C2670	665	21(47)	45(227)	322	14111
C5315	1685	82(204	94(424)	7866	49725
x4	34	5(9)	10(26)	0.8	6.5
industry 1	617	82(278)	238(3025)	81.7	997.9
industry 2	1011	113(324)	322(5574)	165.1	10991.4
industry 3	375	18(39)	141(2129)	20.1	837.0
industry 4	448	61(255)	178(7259)	61.1	2504.3
industry 5	486	61(328)	190(1721)	50.5	1287.6

9 Reference

[1] C. L. Berman and L. H. Trevillyan, “Global Flow Optimization in Automatic Logic Design,” IEEE Trans. CAD 10, pp. 557-564(May 1991).

[2] B. Britton, D. Hill, W. Oswald, N-S Woo, S. Singh, “Optimized Reconfigurable Cell Array Architecture for High-Performance Field Programmable Gate Arrays,” Proc. of Custom Integrated Circuit Conference 1993.

[3] S.-C. Chang and M. Marek-Sadowska, “Technology Mapping via Transformations of Function Graphs”, *Proc. IEEE International Conference on Computer Design*, pp. 159-162, 1992.

[4] J. Cong, Y. Ding, “FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs,” IEEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, No. 1, Jan. 1994, pp. 1-12.

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

[6] L. A. Entrena and K.-T. Cheng, “Sequential Logic Optimization By Redundancy Addition And Removal”, *Proc. International Conference on Computer-Aided Design*, pp. 310-315, Nov. 1993.

[7] R.J.Francis, J.Rose and Z.Vranesic, “Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs”, *Proc. 28th Design Automation Conf.*, pp. 227-233, June 1991.

In our second experiment, we linked our synthesis technique with the router mentioned above for ORCA FPGA. Two unroutable circuits were studied. The first circuit has 169 5-input lookup tables and 738 external wires. Among these external wires, 121 wires have single-wire alternatives. If we run the router using the timing-driven option, the router failed to connect 4 wires even after using the rip-up-and-re-route feature. We have also tried rerouting these 4 wires manually but still not successful because each of these wires has to be routed through some highly congested areas. We then applied our technique to this circuit and successfully completed the routing. The alternative wires of these unroutable wires are reasonably distant from the original wires and can be routed automatically by the router through much less congested areas. The second circuit has 495 5-input lookup tables and 1980 external wires. Totally 1115 wires have single-wire alternatives. There was one unroutable wire in the original design. Again, attempts such as rip-up-and-reroute and manual rerouting were made while not successful. By replacing the wire by its single-wire alternative, the routing was successfully completed. It should be mentioned that this technique is particularly useful when the number of unroutable wires is relatively small. If there are too many unroutable wires, the chance of having a routable alternative for each unroutable wire could be low.

8 Conclusion

Due to the limited routing resources in FPGAs, completion of routing for all wires may not be possible for some FPGA designs, if we are not allowed to change the logic structures. In this paper, we proposed a layout driven synthesis approach that can efficiently identify the alternative wires and/or alternative functions for those unroutable wires. If the alternative wires can be routed through a less congested area, the probability of successful routing will be increased. Our experimental results demonstrate the usefulness of the proposed technique.

TABLE 1 Experimental results

Circuit	# of wires	1-w alt.	3-w alt	cpu 1-w (sec)	cpu 3-w (sec)
f51m	136	25(48)	63(625)	68.1	1222.0
frg1	171	39(102)	85(439)	75.5	1151.2
apex6	854	139(321)	336(953)	348	4434.9
apex7	233	41(72)	75(305)	82.9	563.2
b9	193	28(37)	97(559)	29.0	244.4
misex1	47	8(28)	28(234)	3.0	40.4
misex2	122	53(84)	77(305)	43.0	218.9

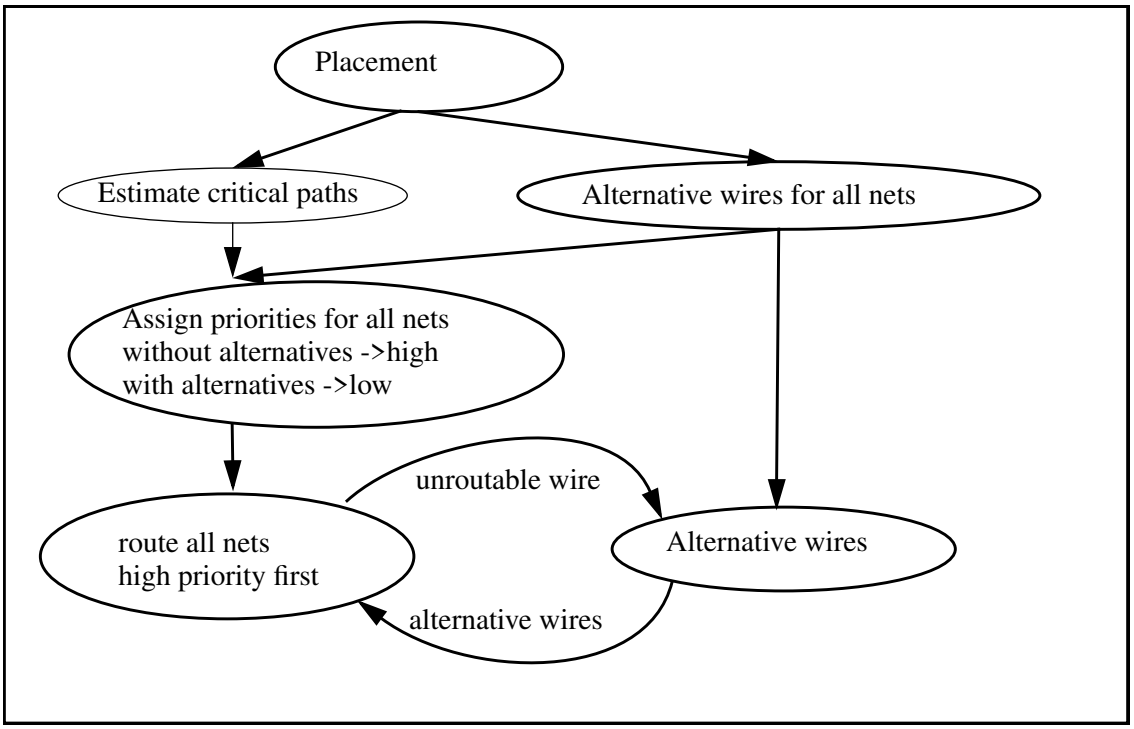


Fig. 14 The flow of layout driven synthesis

7 Experimental Results

We have performed two sets of experiments to demonstrate usefulness of the above approach. The first experiment is to obtain the percentage of wires having alternatives in a circuit. In this experiment, we assume a 5-input lookup table architecture. A circuit is optimized to 5 input lookup tables first. For MCNC examples, we used the SIS script [10]. For industry examples, we used the technology mapper of ODS (ORCA Design System), an industrial tool for AT&T ORCA FPGA [2]. We decomposed each lookup table into simple AND and OR gates. For each external wire (connecting two lookup tables), we search for multiple-wire alternatives. In Table 1, the second column shows the number of external wires in each circuit. The third column shows the number of external wires that have single-wire alternatives; the number in parenthesis shows the total number of alternatives for single-wire alternatives. The fourth column shows the number of wires that have triple-wire alternatives; the number in parenthesis shows the total number of triple-wire alternatives. The cost of an alternative wire is defined in Section 4.4. The fifth and sixth columns show CPU time running on a SUN SPARC 10. We have also measured the number of dominators for each wire. For the set of benchmarks given in Table 1, the average number of dominators for wires in a circuit varies from 2.8 to 6.8.

The overall algorithm for finding an alternative function of a target wire is summarized in Fig. 13.

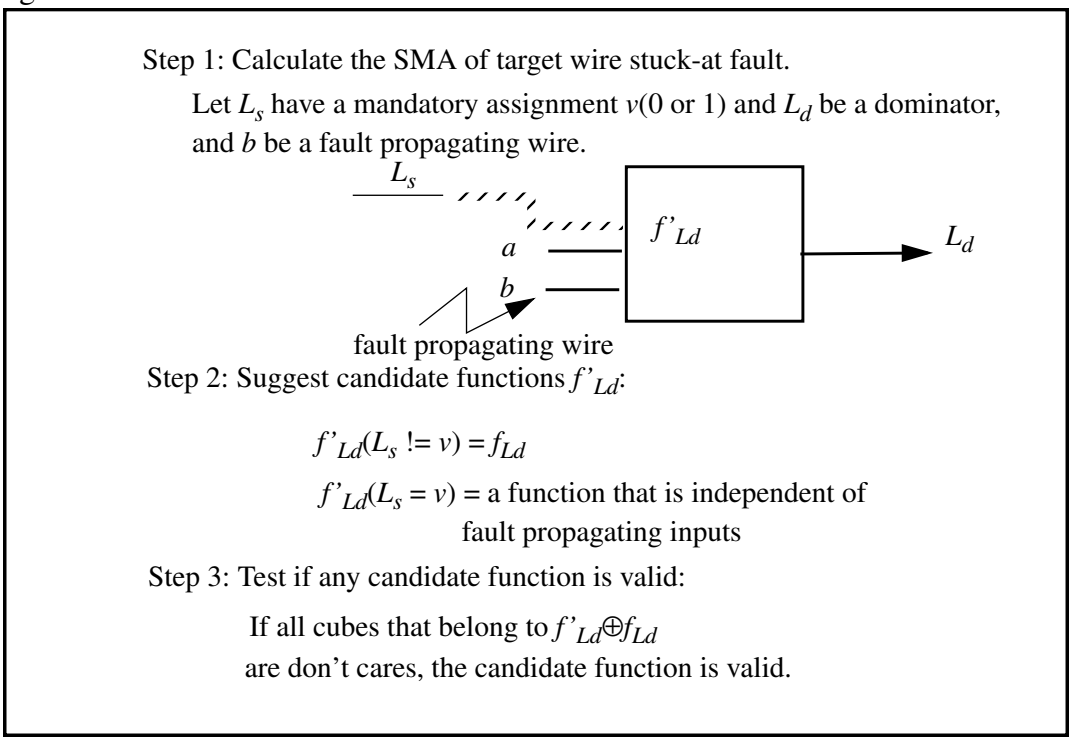
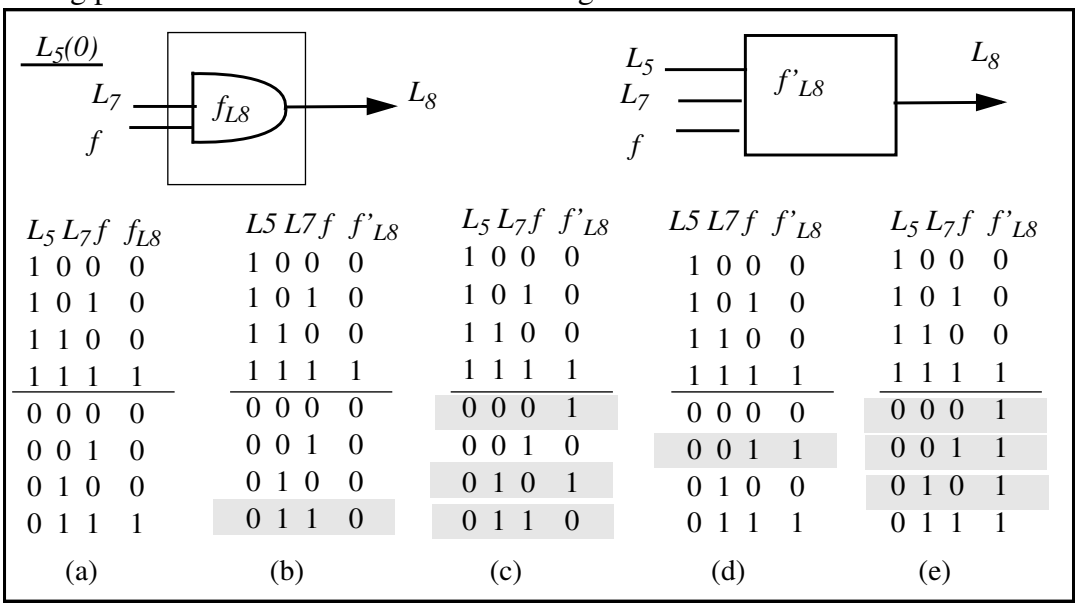


Fig. 13 An algorithm to find alternative functions of a target wire

6 Integration of routing and synthesis

To fully utilize the alternative wire technique, we precompute the alternative wire information and pass it to a routing tool before the routing starts. According to the alternative wire information and the timing information (obtained from other tools), a router sets a priority for each wire. It routes the wires without alternative wires (and time-critical wires) first. This arrangement makes unroutable wires more likely to have alternatives. If a wire cannot be routed and it has alternatives, the router tries to route those alternatives one-by-one. If all alternative wires of an unroutable wire cannot be routed, we also try ripping up some wires around the congested area. Fig. 14 shows a design flow of the layout driven synthesis process.

1, 1) can be assigned to either 0 or 1 and the function of the circuit does not change. The following procedure shows a method of checking whether a minterm is a don't care.



L_5 L_7 f	f_{L8}
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1

(a)

$L5$ $L7$ f	f'_{L8}
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	0

(b)

L_5 L_7 f	f'_{L8}
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	0

(c)

$L5$ $L7$ f	f'_{L8}
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1

(d)

L_5 L_7 f	f'_{L8}
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1
0 0 0	1
0 0 1	1
0 1 0	1
0 1 1	1

(e)

Fig. 12 An example of finding alternative functions

Suppose we try to verify whether $(v_1, v_2,...,v_n)$ is a don't care cube of a function $f_{Lm}(L_1,L_2, ..., L_k)$. This cube is first set as mandatory assignments to the inputs of L_m. Then, we try to sensitize a path from L_m to a primary output. If it is not possible to simultaneously justify the cube and sensitize a path from L_m to any primary output, the cube $(L_1,L_2, ..., L_n) = (v_1,v_2,...,v_n)$ is a don't care of $f_{Lm}(L_1,L_2, ..., L_n)$.

As an example, let us check if the cube $(L_5, L_7, f) = (0, 1, 1)$ in L_8 of Fig. 11 is a don't care. We first set L_5=0, L_7=1, f=1 and propagate a fault from L_8 to primary outputs. Then, we calculate the mandatory assignments as described in section 3. Since the mandatory assignments conflict, $(L_5, L_7, f) = (0, 1, 1)$ is a don't cares for L_8.

Lemma: If we replace f_{Li} by f'_{Li}, a candidate function derived from the candidate_searching procedure, the SMA of the target wire stuck-at-fault will be inconsistent.

Proof. Since L_i is a dominator of the target wire in its stuck-at fault test, we must be able to observe the target fault at L_i's output. Suppose the function in L_i is f'_{Li}. We must set x_n to the value $!v$ to observe the fault. It is because when x_n has the value v, f'_{Li} is independent of all the fault propagating wires in X_1. In addition, when x_n has the value $!v$, the above procedure sets f'_{Li} as the same as f_{Li}. Since we connect the input pin x_n from the wire that has the mandatory value v, x_n has value v. Thus we get a conflict of assignment at x_n. Therefore, if f'_{Li} is the function in the lookup table L_i, the target wire is untestable. Q.E.D.

For example, in Fig. 11, we would like to find the candidate functions in the lookup table L_8 to remove the target wire L_1->L_4. According to the procedure, we may use L_5, which has mandatory assignment 0, as the extra-input of L_8. The objective of the candidate_searching procedure is to find Boolean functions in the lookup table L_8 that can make L_1->L_4 redundant. Fig. 12(b,c,d,e) shows 4 possible candidate functions. When L_5 =1 the four functions are the same as the original function ($L_7 \bullet f$); when L_5 =0 the functions are independent of L_7 which is in the fault propagating path. If any of the Boolean functions in Fig. 12(b,c,d,e) replaces the Boolean function in L_8, the target wire L_1->L_4 becomes redundant.

5.3 Verifying validity of a candidate function

After suggesting candidate functions of some lookup table to remove a target wire, we still need to verify whether a candidate function is redundant or not. For example, after obtaining the four candidate Boolean functions in Fig. 12, we need to check whether they are valid. First, we compare the old function with each of candidate functions. For example, in Fig. 12, the discrepancy between the original function (a) and function (b) occurs in the minterm (L_5, L_7, f) =(0, 1, 1) as highlighted in Fig 10(b). In order to maintain the circuit's functionality, the minterm (0, 1, 1) must be a "don't care" for the circuit, i.e. f'_{L8}(0,

therefore, it is an alternative function for L_1->L_4. We call the new input to the lookup table L_8, L_5, an *extra-input* of the alternative function.

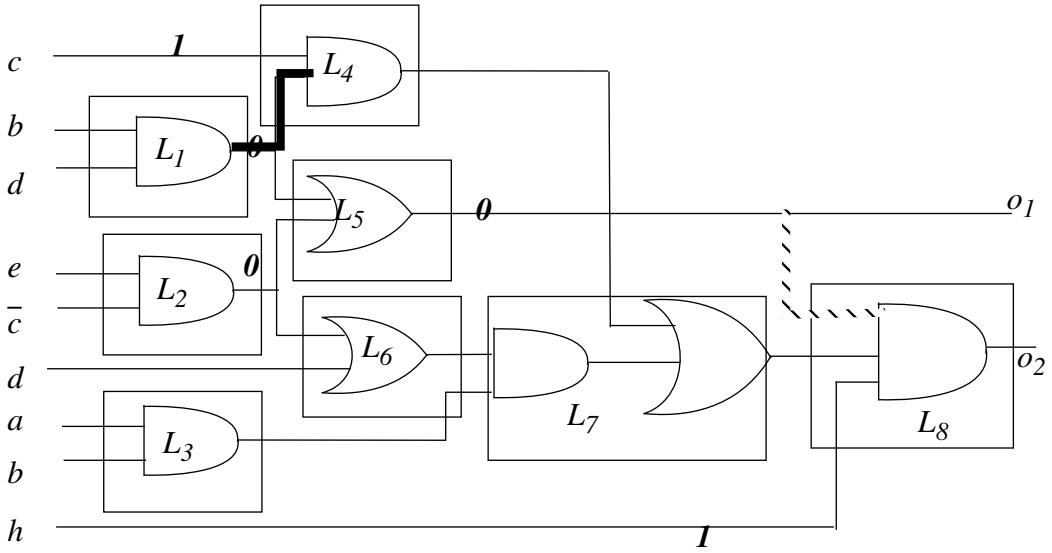


Fig. 11 Example of alternative function

5.2 Deriving candidate functions

A lookup table that we consider for functionality change must satisfy two conditions: (1) the lookup table is a dominator of the target wire and (2) the lookup table has at least one unused input, or the lookup table has no unused inputs but it has an input with a mandatory assignment. When the lookup table has at least one unused input, we may connect to it a wire, an extra-input, which has a mandatory assignment. In the example of Fig. 11, L_8 is one of the dominators and the extra-input L_5 has a mandatory value 0. Suppose a lookup table L_i satisfies the above two conditions. In the following, we describe a procedure to derive candidate functions for the lookup table L_i. We refer this procedure as the *candidate_searching* procedure. Suppose the inputs to L_i are X =(X_1, X_2) and X_1 contains inputs in the transitive fanout of the target wire.

After calculating the mandatory assignments of the target wire stuck-at fault, we identify L_i's input pin x_n which has a mandatory assignment *v*. If there is no input of L_i with mandatory assignment, we do the following. We identify a wire which has a mandatory assignment *v. We* will connect the wire to a new input pin of L_i; let the new input pin also named x_n. Now, we define a candidate function f'_{Li} by specifying a new truth table as follows. When x_n has the value *!v*, we set f'_{Li} to be the same as f_{Li}; when x_n has the value *v*, we set f'_{Li} as any function independent of X_1. (Recall that X_1 is a set of input variables, which are in the transitive fanout of the target wire, of L_i.)

5 Using alternative functions of lookup tables to remove a target wire.

We now show how to generalize the alternative wire technique for lookup table (LUT) based FPGAs. We introduce the notion of alternative functions of an unroutable wire. This generalized technique operates directly on the LUT level without decomposing the mapped LUT into primitive gates.

5.1 Alternative functions of lookup tables

A k-input LUT can realize any of 2^{2^k} Boolean functions. By replacing the Boolean function of a LUT by another one among these 2^{2^k}functions, some wires may become redundant.

Let L_o be a lookup table in a circuit and $f_{Lo}(X)$ be its Boolean function where X is a set of inputs to L_o. We denote the cardinality of X as $|X|$. Therefore, |X| <= k. We say $f'_{Lo}(X')$ is a *valid* function if (1) replacing $f_{Lo}(X)$ with $f'_{Lo}(X')$ does not change the functionality of the circuit and (2) $|X'|$ <= k. Given a target wire w to be removed, our objective is to find a valid function of some lookup table such that w becomes redundant. We call such a valid function an *alternative function* of the target wire w.

The process of finding alternative functions consists of three steps. First, we calculate the SMA of the target wire stuck-at fault. Second, a set of candidate functions is generated. The procedure guarantees that each of these candidate functions makes the target wire redundant. Finally, we check whether these candidate functions are valid. Consider the circuit in Fig. 11 where each box represents a LUT. Let L_1->L_4 be the target wire. First we calculate the SMA of L_1->L_4 s-a-1; SMA values are shown in the figure. In Step 2, the procedure suggests that $f'_{L8}(L_5, L_7, h) = L_5 \bullet L_7 \bullet h$ (the original function is $f_{L8}(L_5, L_7, h) = h \bullet L_7$) is a candidate function (how this candidate function is derived will be explained in the following paragraphs). Finally, we verify that this candidate function is valid and,

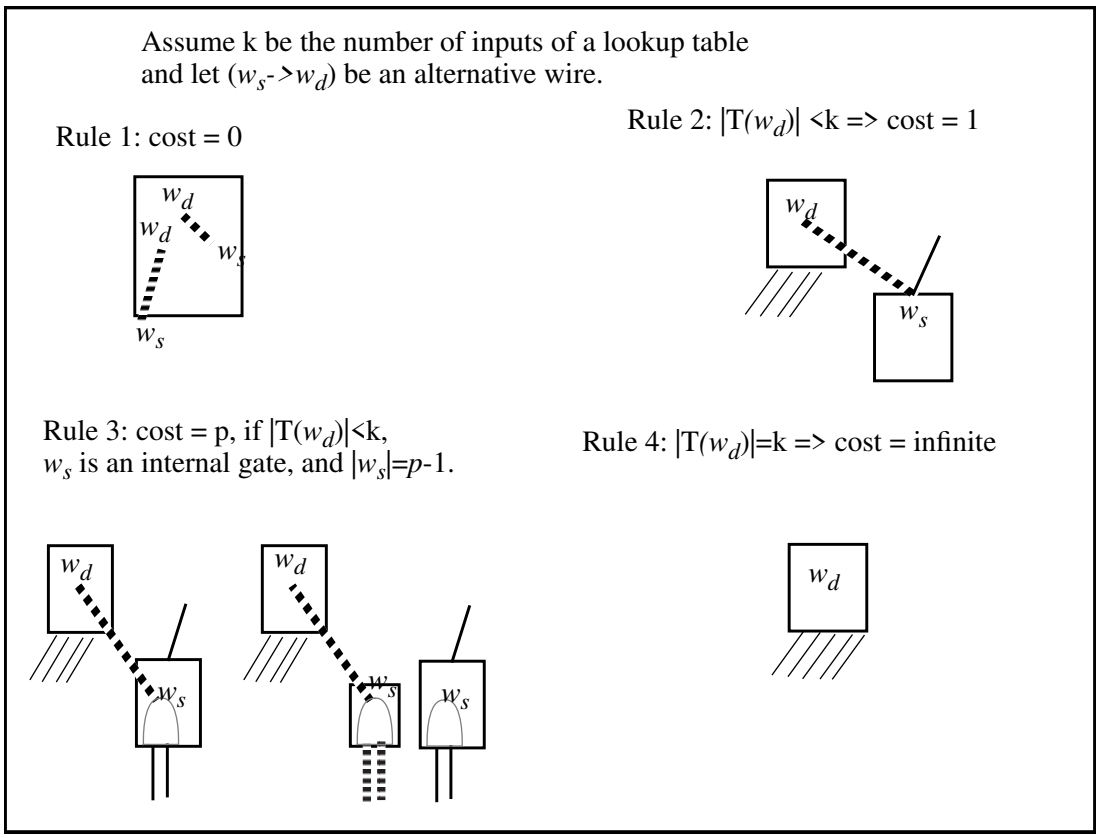


Fig. 10 Cost of alternative wires

Let each alternative wire be expressed as (w_s, w_d) where w_s and w_d represents gates. We assign a cost to an alternative wire based on the following rules. These rules are used to calculate the number of external wires to be added to the circuit for the corresponding transformation. Fig. 10 summarizes these rules. We use T(w_s) to denote the lookup table containing gate w_s and |T(w_s)| to denote the number of inputs of T(w_s). Of the inputs to T(w_s), let |w_s| be the number in the transitive fanin of w_s.

- Rule 1: cost =0, if T(w_s) = T(w_d).
- Rule 2: cost =1, if T(w_s) ≠ T(w_d), w_s is the output of T(w_s) and T(w_d) has less than k inputs.
- Rule 3: cost = p, if T(w_s) ≠ T(w_d), w_s is not the output of T(w_s), |w_s|= p-1 and T(w_d) has less than k inputs. We need to construct an extra lookup table to generates w_s.
- Rule 4: cost = infinite, if T(w_s) ≠ T(w_d) and T(w_d) has k inputs. Note that we can use Shannon decomposition to break T(w_d) into smaller lookup tables but we choose not to consider this substitution.

behavior of the network. Using such summary information, one signal net can be restructured. For example, Fig. 9a shows a signal net with three wires s->d_0, s->d_1 and s->d_2. After gathering the summary information, the data flow technique may conclude that by adding s->d_3 and s->d_4, s->d_0, s->d_1 and s->d_2 can be removed. Such a net restructuring can be considered as a restricted way of finding alternative wires. In this example, if we want to remove s->d_0, this technique would report a solution by adding s->d_3 and s->d_4 and removing s->d_1 and s->d_2. In the context of finding alternative wires, our technique is more general compared to the flow technique. It is because the flow technique adds or deletes only the wires coming from the same source node, while our technique adds wires originating from any node. In other words, our technique can find *indirect* alternatives that do not share source and destination gates of the target wires and, thus, are likely to be more distant from the wires desired to be removed. Also, the use of several ATPG-based techniques makes our method more memory and CPU efficient than the global flow method. On the other hand, the flow technique has the advantage of adding and removing several wires simultaneously.

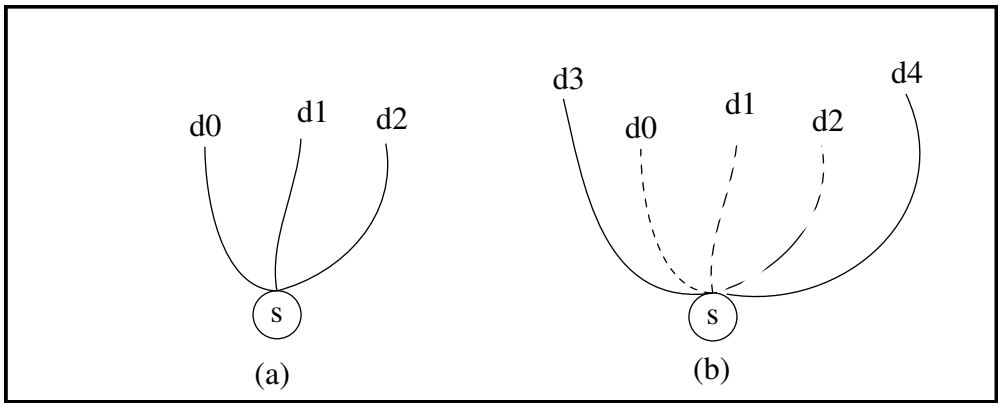


Fig. 9 An example of global flow

4.4 Complex gates

To apply our technique to lookup table architecture, we decompose each lookup table into AND, NAND, OR and NOR gates. After decomposition, we have two kinds of wires in the circuit. An *external* wire connects two gates that belong to two different lookup tables. An *internal* wire connects two gates that belong to the same lookup table. For an external target wire, we can obtain a set of multiple-wire alternatives using the approach described earlier. In the following, we discuss the *cost* of an alternative wire.

of g_5->g_9. Because g_5->g_9 is a redundant wire, we can add g_5->g_9 and, thus, add g_3->g_8 to the circuit. Finally, we find g_7->g_8 is a redundant wire so it can be removed.

4.3 Comparison with other related work

In the following, we discuss two related previous works which can be modified to obtain the same purpose as ours. The first set of works is the transduction method from [11]. Using the concept of permissible functions, they have developed a procedure for error compensation. The goal of this procedure is to add wires/gates to compensate the errors created by removing a gate/wire in a circuit. Though this error compensation procedure was designed for logic optimization, in fact, this procedure can also find alternative wires for a target wire. We first briefly outline their idea and then, we make a comparison between our and their techniques.

This error compensation procedure consists of three steps. In the first step, a wire/gate is removed from the network. Let a primary input vector *v* be an *error component* if evaluating *v* can distinguish between the original circuit and the circuit after removing the wire/gate. Then, permissible functions with errors for each nodes throughout the network are calculated. These permissible functions with errors represent functions with error components as well as ordinary permissible functions that have no erroneous components. Finally, an attempt is made to compensate errors by adding wires/gates. Such a procedure is very time and memory expensive. Computing and expressing permissible functions (in terms of primary inputs) for all nodes in a circuit requires enormous cpu time and memory. In addition, to our understanding, there is no proper guidance to add multiple wires/gates for error compensation. Once a wire is replaced by another wire, the permissible functions for all the nodes need to be recalculated. Such recalculation of permissible functions is very costly. On the other hand, the use of the concepts of dominators and mandatory assignments makes our approach both memory and CPU efficient. We have techniques to find both single and multiple wires/gates for error compensation. Also, our technique only keeps minimum global information. The only information updated globally after replacing a wire is the information about nodes' dominators. One can compute the dominators of all nodes in the worst case of n^2 where n is the number of nodes in a circuit.

The other approach which can be modified for finding alternative wires is described in [1]. In the paper, Berman & Trevillyan proposed a data flow analysis technique. First, a summary information about the circuit is gathered in the form of implicants which result in identification of wires to be connected or disconnected without affecting the external

is not an alternative wire. However, we can try to add other wires to make an irredundant candidate connection redundant.

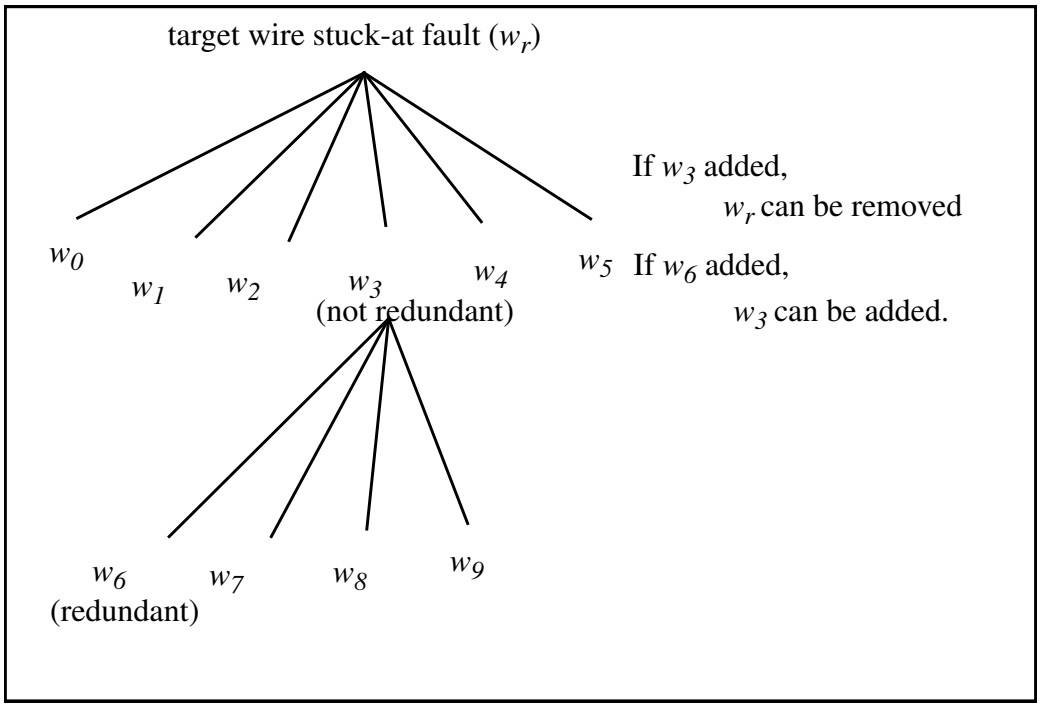


Fig. 8 Identifying multiple-wire alternatives

Fig. 8 shows a search tree to find the alternative wires for w_r. Suppose there are 6 candidate connections, w_0,..., w_5, suggested by the procedure of single-wire alternatives and none of them is redundant. We then try to add other wires to make the candidate connections redundant. Each candidate connection wire is considered as a new target wire. In our example, we consider wire w_3 as a new target wire. The same procedure of finding single-wire alternative is applied now for w_3. Let wire w_6, w_7, w_8 and w_9 be the new candidate connections for making w_3 redundant. If wire w_6 is a redundant wire, we conclude that we can add wire w_6 so that wire w_3 can be added to the circuit without changing the circuit functionality. Finally, we add wire w_3 and wire w_6 in the circuit and verify whether w_r is still a redundant wire. The reason that the removal of w_r needs additional verification is explained below.

In searching for the single-wire alternatives, we know that the addition of wire w_3 guarantees the removal of wire w_r and the addition of wire w_6 guarantees the addition of wire w_3. However, the addition of both wire w_3 and w_6 does not necessarily guarantee the removal of w_r although most of the time it is the case. For example, in Fig. 1, the removal of g_7->g_8, suggests the addition of g_3->g_8. The addition of g_3->g_8 suggests the addition

tory value 0. As a result, after the Type 2 transformation is performed, the SMA of the target wire becomes inconsistent and the target fault becomes untestable. In these four transformations, the gate g_d is assumed to be an AND gate. We can also derive similar transformations for the case that the dominator is an OR gate.

Fig. 7 shows a pseudo code for finding all single-wire alternatives of a target wire. For each candidate connection (g_s->g_d), we consider all types of circuit transformations suggested in Fig. 6.

```
single_wire_alternative(wire, stuck_type) {
        compute_SMA(wire, stuck_type);
        for (each gate g_i in the circuit){
            if (g_i has mandatory assignment) insert g_i into the source_list.
            if (g_i is a dominator)           insert g_i into the destination_list.
        }
        for (each g_s in the source_list) {
            for (each g_d in the destination_list) {
                if (gs is in the fanout of gd) continue;
                if(verify_redundant(g_s, g_d, type_0or 1))
                    insert_into_alternative_wire_list(g_s, g_d, type_0or1);
                if(verify_redundant(g_s, g_d, type_2or3))
                    insert_into_alternative_wire_list(g_s, g_d, type_2or3);
            }
        }
}
```

Fig. 7 Pseudo code for finding single-wire alternative

4.2 Multiple-Wire Alternatives

Sometimes a wire does not have any single-wire alternative. In this section, we describe a method how to add several wires/gates to remove a target wire.

Fig. 8 shows the search process for a multiple-wire alternative of a target wire. Let wire w_r be the wire to be removed. We first try the three-step single wire procedure in Fig. 5. After computing the mandatory assignments of the target wire stuck-at fault, the procedure suggests a set of candidate connections. If a candidate connection is not redundant, it

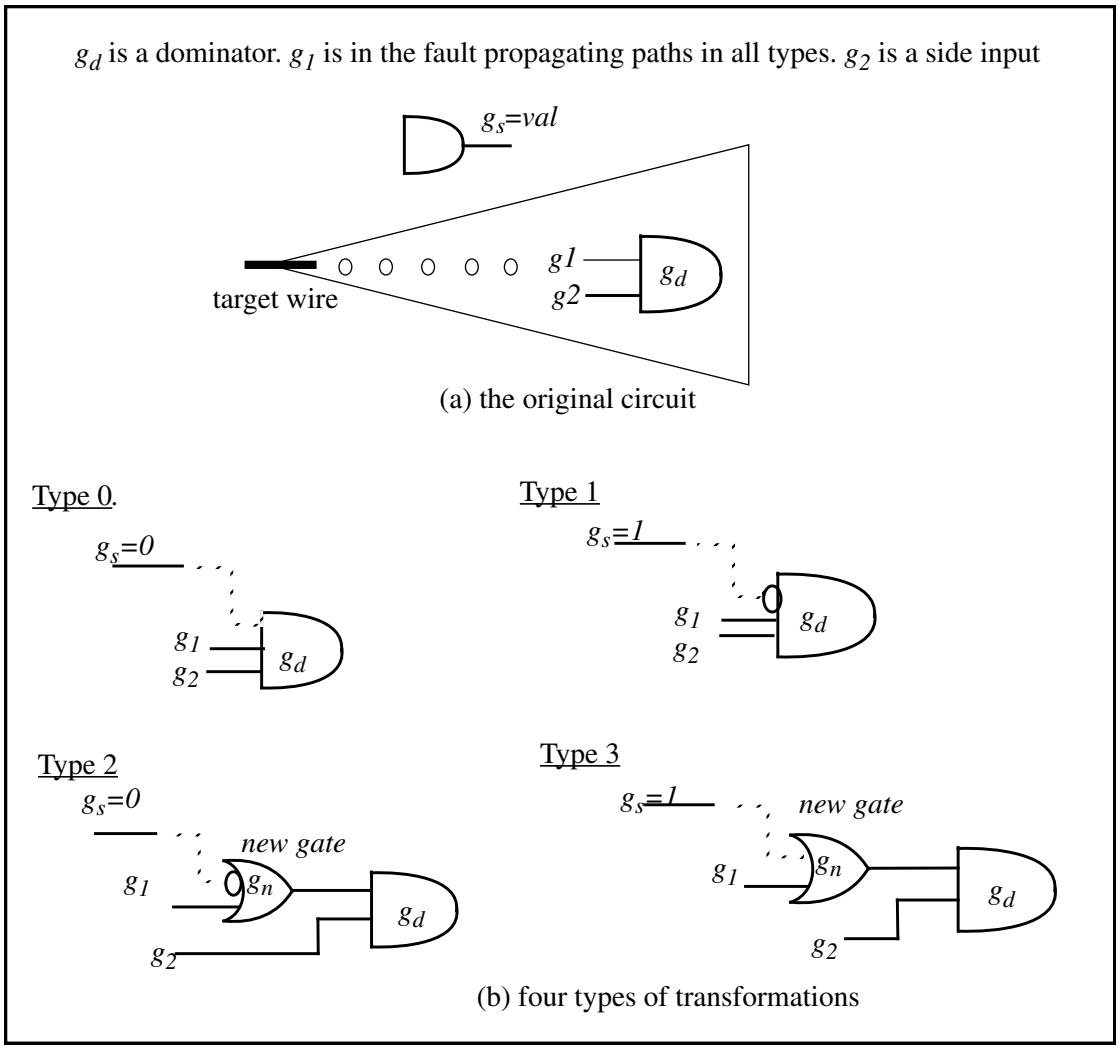


Fig. 6 Types of transformations

All candidate connections (g_s g_d) in Fig. 6 use the same principle described as follows. In Fig. 6a, the destination gate g_d is a dominator of the target wire. The source gate g_s has a mandatory assignment and we also restrict that g_s is not in the transitive fanout of the target wire. This is because if g_s is in the transitive fanout of the target wire, adding g_s->g_d may alter the g_s's mandatory assignment. Type 0 and Type 1 in Fig. 6b add a wire from g_s to the dominator g_d. Type 2 and Type 3 add a new gate g_n first; then, reconnect all fault propagating wires previously connected to g_d to g_n and connect a wire from g_s to g_n.

We will discuss Type 2 transformation below; the same argument can be used for Type 0, 1 and 3 transformations. Suppose g_1 is a fault propagating wire. Because g_d is a dominator, the new added gate g_n is also a dominator. Since g_s is a side input to the dominator g_n, g_s must be assigned to a non-controlling value, 1, which conflicts the original manda-

may change the circuit's behavior. Therefore, in the final step, we need to check whether a candidate connection is redundant or not. If a candidate connection is redundant, we conclude that the candidate connection is an alternative wire for the target wire. Note that in this step, the redundancy checking is performed with the target wire in the circuit. The following example illustrates the process of finding single-wire alternatives.

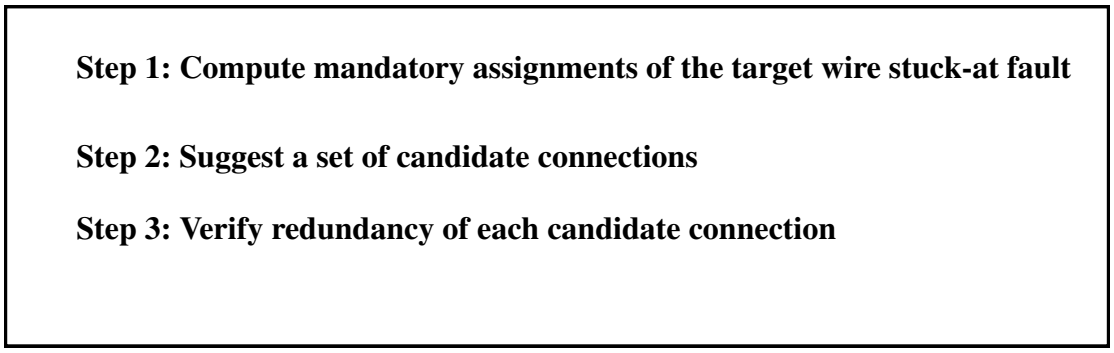


Fig. 5 The procedure of finding alternative wires

Example: Consider the circuit in Fig. 3. Let g_1->g_4 be the target wire to remove. First, we compute the SMA(g_1->g_4 stuck-at-1). We have SMA = {c=1, g_1=0, g_5=0, g_2=0, g_7=0, f=1}. Note that g_5 is outside the transitive fanouts of the target wire and has a mandatory value 0. If we connect g_5 to g_9, a dominator, it will cause inconsistency in mandatory assignment for g_5. This is because if the new wire g_5->g_9 is added, g_5 becomes a side input of the dominator g_9. It requires a non-controlling value **1** at g_5 to propagate the fault effect of the target fault to the outputs. This additional requirement causes conflict in the SMA. The process, thus, suggests wire g_5->g_9 as a candidate connection. Finally, we check whether g_5->g_9 is redundant by examining the consistency of the SMA for the s-a-1 fault at wire g_5->g_9. The SMA of this fault is inconsistent as discussed in Section 3 and therefore, wire g_5->g_9 is an alternative wire for wire g_1 ->g_4.

In the following we discuss details of Step 2. Given a target wire to remove and assume the dominator is an AND gate, Fig. 6 summarizes four different types of candidate transformations in Step 2. After any of these four transformations, the SMA of the target wire becomes inconsistent and the target wire becomes redundant. The dotted wires in Fig. 6 are the candidate connections. We say a wire w is a *fault propagating wire* if there is a path from the target wire to w.

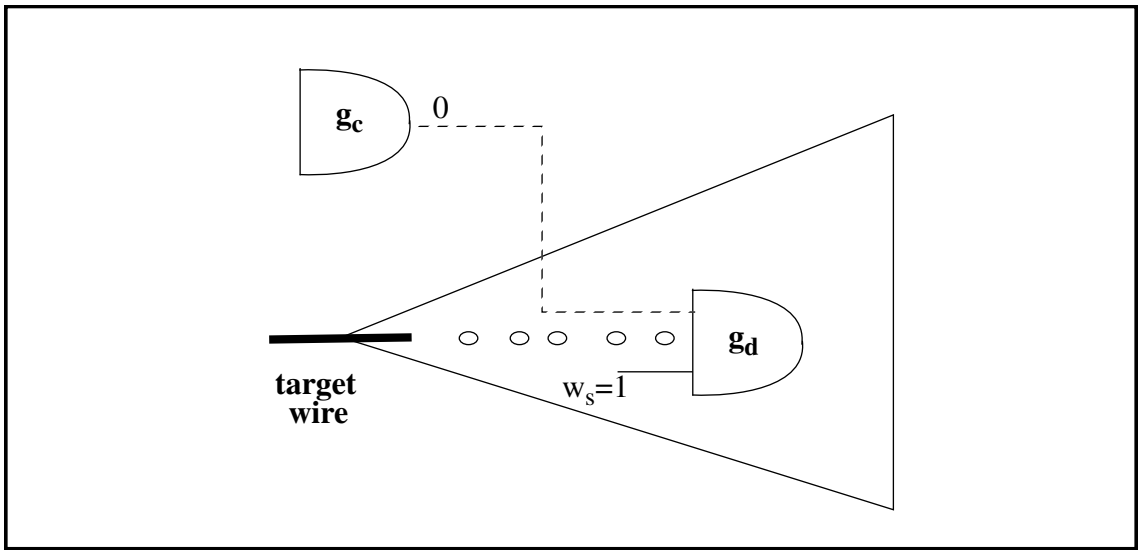


Fig. 4 The addition of a wire may cause a target wire stuck-at fault untestable

4 The alternative wires of a target wire

In this section, we will show a method of adding alternative wires to make a target wire redundant. Basically, this method will find and add redundant wires to cause the target fault untestable and therefore the target wire redundant. The idea was originally proposed in [5] [6], in which their objective was to achieve logic optimization by adding one redundant wire to remove other redundant wires. On the other hand, our objective is to remove some target (i.e., unroutable) wires by adding one or more wires and/or gates. Therefore, we allow adding several wires/gates and changing a gate's function to remove the target wires. In the following, we first show how to add a single wire/gate to remove a target wire. Then, we discuss how to add multiple wires/gates to remove a target wire. After that, we discuss some related previous work. Finally, we discuss the cost of adding a wire in the LUT type FPGA.

4.1 Single-wire alternative

This section describes a procedure that finds single-wire alternatives of a target wire. The procedure consists of three steps shown in Fig. 5. In the first step, we calculate the mandatory assignments of the target wire stuck-at-fault. Then, we identify a set of candidate connection wires to be added. As we explained adding the dotted wire in Fig. 4, each candidate connection, when added to the circuit, will cause the stuck-at fault of the target wire untestable and therefore redundant. However, adding such a candidate connection

learning [9]. If the mandatory assignments for a stuck-at fault cannot be consistently justified, the fault is untestable and therefore, the wire is redundant.

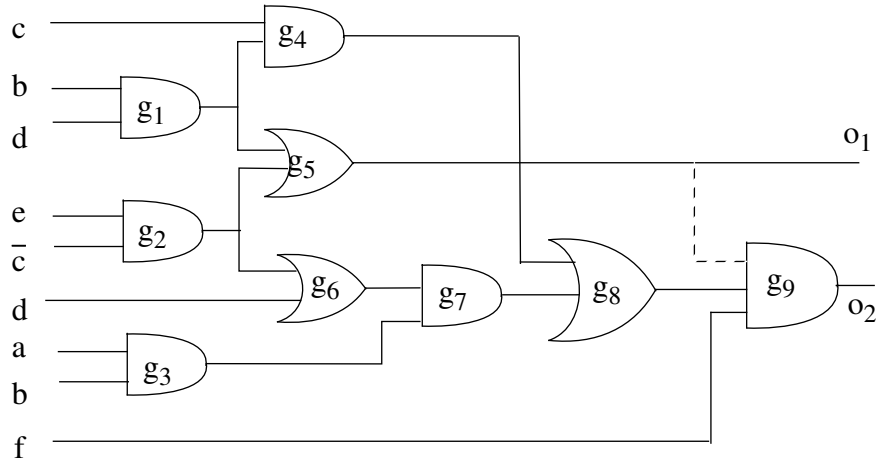


Fig. 3 Example of redundant wire.

For example, we like to know whether the dotted wire g_5->g_9 can be added in Fig. 3. First, we need to know whether g_5->g_9 s-a-1 is testable. The mandatory assignments for g_5->g_9 s-a-1 are g_5=0, g_8=1, f=1, g_1=0, g_2=0, g_4=0, g_7=1, g_6=1, g_3=1, a=1, b=1, d=1, and g_1=1. Because the mandatory assignment of gate g_1 cannot be consistently justified, the g_5->g_9 s-a-1 fault is untestable and therefore wire g_5->g_9 is a redundant wire. Thus, the g_5->g_9 wire can be added.

Let us consider testing the target wire stuck-at fault in Fig. 4 and suppose the dotted wire is not present in the figure. To observe the fault effect at the output, we must sensitize a fault propagating path. Suppose g_d is a dominator of the target wire. The side inputs of g_d must be assigned to the non-controlling value for testing the target fault. For example, w_s must be 1. Suppose another gate g_c outside the transitive fanout of the target wire has a mandatory assignment 0. If the dotted wire is now present in the circuit, the 0 at g_c, the controlling value to g_d, will block the fault propagation of the target wire and make the target fault untestable and therefore redundant. This example shows that the addition of some wire may block the fault propagation of another wire and thus make it redundant.

In this paper, we consider to solve the following problems. Given a wire w_r to remove, we attempt to perform the following transformation to so that w_r becomes redundant. Two nodes n_s, and n_d are identified first. Then, add a wire from n_s to n_d and at the same time may modify the function of node n_d. Note that in this formulation. nodes in a circuit can be complex node. We start with considering circuits with only AND and OR gates in section 3, 4, and then considering complex function in section 5.

3 Redundancy identification & Redundancy Addition

We will review the idea in, [6] [8] that identifies redundancies using the concept of mandatory assignments. Consider the absolute dominators of a wire *W*. Let *side inputs* of a dominator be its inputs not in the transitive fanout of *W*. To generate a test for a stuck-at fault at wire *W*, all side inputs of *W*'s dominators must be assigned to their non-controlling values. For example, in Fig. 3, to test wire (g_1->g_4) s-a-1 (stuck-at-1), we must assign 1 to *c*, 0 to g_7, and 1 to *f*.

When we perform Automatic Test Pattern Generation (ATPG) for a wire stuck-at fault, a test vector must be able to activate the fault and to propagate the fault to at least one primary output. For example, in Fig. 3 to test g_1->g_4 s-a-1, any test vector must satisfy g_1=0 for fault activation and must satisfy {*c=1, g_7=0, f=1*} (side inputs to dominators) for the fault propagation.

The mandatory assignments are the value assignments required for a test to exist and must be satisfied by any test vector. Given a stuck-at fault *f*, we compute the set of mandatory assignments SMA(*f*) [6] that can be computed via implication [8] [12] or recursive

Two circuits with unroutable wires were tried and were successfully routed using our technique.

The remainder of this paper is organized as follows. Section 2 gives some definitions. Section 3 reviews the redundancy identification technique that forms a core of our method. Section 4 details the proposed method of finding alternative wires of a target wire. Section 5 discusses the alternative function for a target wire. Section 6 describes the algorithm for the integrated synthesis and routing process. Section 7 shows some experimental results and conclusion follows in Section 8.

2 Definitions and problem formulation

In this paper, we follow the standard logic synthesis terminology. A Boolean network is a Directed Acyclic Graph(DAG) where each node is associated with a Boolean function f_i, and a Boolean variable y_i. There is an edge directed from a node n_i to a node n_j if the function f_j depends on the variable y_i. The node n_i is a fanin of the node n_j and the node n_j is a fanout of the node n_i. If there is a directed path from a node n_i to a node n_j then we say that n_i is in transitive fanin of n_j and that n_j is in transitive fanout of n_i. We use the term edge and wire interchangeably in this paper.

Definition: An *absolute dominator*, also called dominator [8], of a wire W is a gate G such that all paths from wire W to any primary output have to pass through G. For example, the dominators of g_1->g_4 in Fig. 3 are g_4, g_8, *and* g_9.

Definition: The value of an input to a gate is said to be *controlling* if it determines the value of the gate's output regardless of the values of the other inputs; the controlling value is 1 for an OR or a NOR gate, and 0 for an AND or a NAND gate. The inverse of the controlling value is called the non-controlling value or sensitizing value.

A wire to be removed is referred to as the target wire. The corresponding stuck-at fault is called the target fault. *Alternative* wire(s) of a target wire make the target wire redundant when they are added to the circuit. In a combinational circuit, a wire is redundant if and only if the corresponding stuck-at fault is untestable.

We say that a wire w_1 is a single-wire alternative of a target wire if w_1 can replace the target wire.

The interconnection wires between chips typically cause long propagation delays. If an interconnection wire between chips has alternative wires all within chips, it may be replaced by them to reduce the critical path delay. Yet another application is shown in Fig. 2c. Suppose the pin constraint of a chip is violated after a circuit is partitioned. If an interconnection wire has alternative wires all inside chips, we may reduce the number of interconnection wires by using its (internal) alternative wires. All the above applications explore the alternatives in the logic domain to improve results in physical design. Note that, in addition to FPGAs, the technique can be applied to standard cell and gate array designs as well.

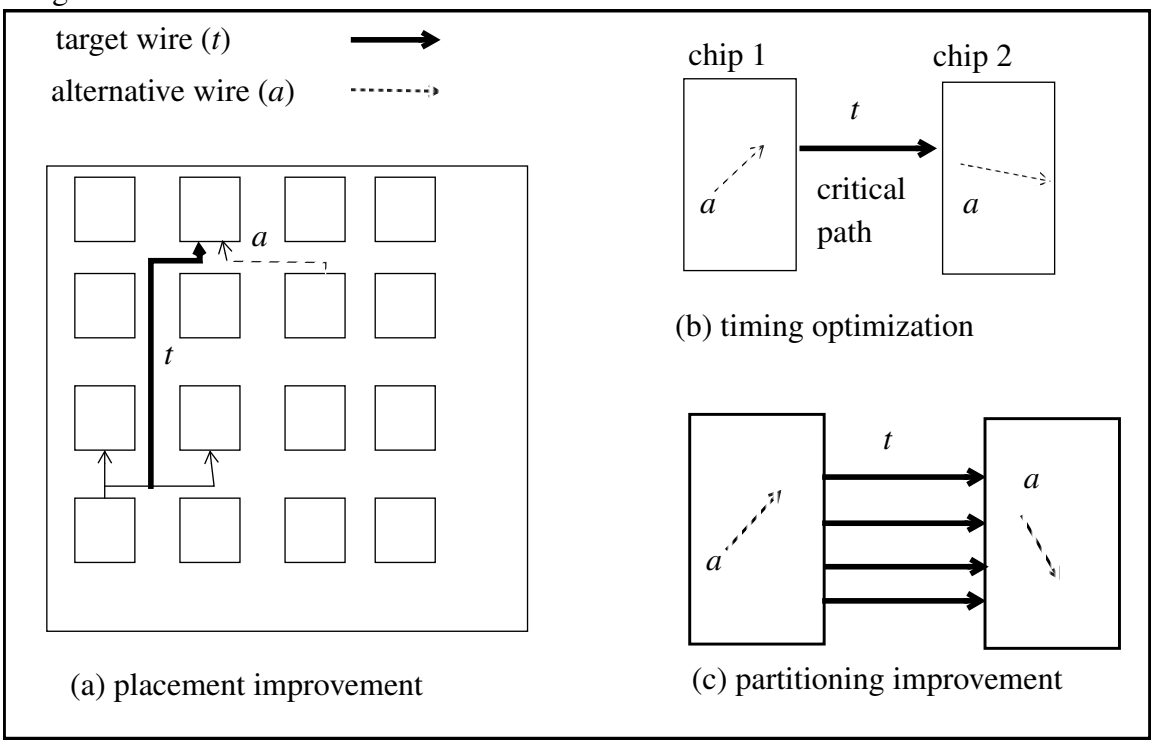


Fig. 2 Applications of alternative wires

We have developed an efficient method to find alternative wires of a target wire. The method is an extension of the redundancy addition and removal techniques [5] [6]. Two experiments were performed to demonstrate the usefulness of this technique. In the first experiment, our goal was to find the percentage of wires that have alternative wires. We first mapped several randomly selected MCNC benchmarks and industry examples to 5-input lookup tables. For each wire in the circuit, we checked if there exist alternative wires. We say that a wire has *a triple-wire* alternatives if it can be replaced with 3 or fewer wires. The experiment showed that 30%-50% of wires have triple-wire alternatives. In the second experiment, we linked our synthesis tool to a router for AT&T ORCA FPGA [2].

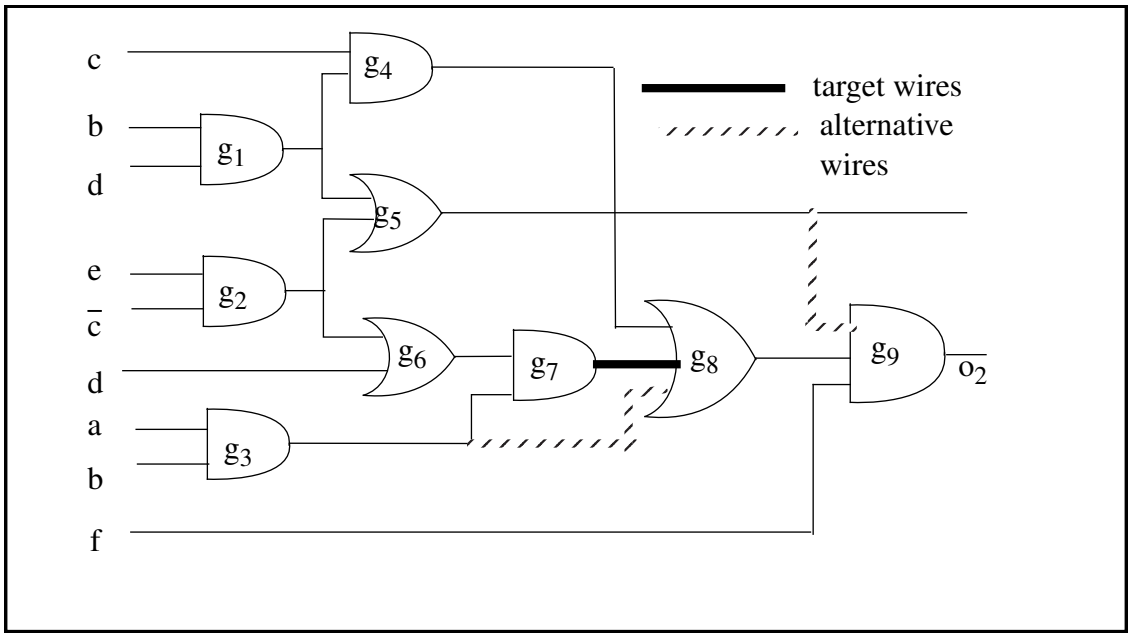


Fig. 1 Example of alterative wires

For example, in the circuit depicted in Figure 1 (from [5]), we can remove wire g_7->g_8 by simultaneously adding wires g_3->g_8 and g_5->g_9. In our technique, we also allow adding gates and/or changing the functions of logic blocks in the circuit when removing target wires. Note that for lookup table-based FPGAs, adding a gate or a wire inside a lookup table can be done by simply changing the truth table of the lookup table.

The use of alternative wires to replace unroutable wires gives routing an additional flexibility. A closer integration of this technique with routing can dramatically improve the routability of FPGAs. We assign a higher priority to wires that do not have alternatives and instruct the router to route high priority wires first. If routing cannot be completed, the unrouted wires are more likely to have alternative wires. The router then interacts with the synthesis program to obtain their alternatives. If any of such alternatives can be routed through a non-congested area, that alternative will be used and routed. The process continues until all unrouted wires are replaced by their routable alternatives.

This alternative-wire technique has other applications as well. Consider Fig. 2a. After placement, the routing length of wires can be estimated. If an estimated long wire (target wire *t* in Fig. 2a) has a shorter alternative wire (wire *a* in Fig. 2a), the long wire can be replaced by the short wire to improve both area and performance. Based on the same principle, we can apply this technique for timing optimization, as already pointed out in [5]. For another example, a circuit shown in Fig. 2b is partitioned and placed on two chips.

1 Introduction

Field programmable gate array (FPGA) consists of an array of identical logic blocks and routing resources. In a LookUp Table (LUT) architecture, each logic block can implement any k-input Boolean function where k is decided by the size of the lookup table.

The traditional design flow for FPGAs consists of four steps. In the first step, a logic optimizer performs technology independent optimization on a circuit. Then, a technology mapper (e.g., [3] [4] [7] [10] [13]) maps the circuit onto logic blocks to minimize the number of lookup tables used and/or to minimize delay. Placement and routing then follow. During the physical design phase, the logic information of the circuit is no longer used. Only the topology, typically represented as a graph, is retained. Because of the limited routing resources available in FPGAs, routing may fail in the congested area, even though there are routing resources available elsewhere. When the routing fails, there is no systematic way to introduce incremental changes and complete the routing. To make an unroutable circuit routable, a user may change the placement by swapping or duplicating some logic blocks to alter the topology of the circuit. Such changes, however, lead to unpredictable results and quite often it is not clear how to alter the design. In this paper, we propose a layout driven synthesis technique that performs incremental transformations in the logic level (without changing the circuit's functionality) to improve the routability of the FPGA. According to the information fed back from the routing tool, the wiring topology is altered appropriately to complete the routing.

Our synthesis method looks for *alternative wires* for each wire that cannot be routed. The alternative wires of a target wire are wires which can replace the target wire without changing the circuit's functionality.

Post-layout Logic Restructuring Using Alternative Wires

Shih-Chieh Chang, Kwang-Ting Cheng, Nam-Sung Woo and Malgorzata Marek-Sadowska

Abstract

In this paper, we propose a layout driven synthesis approach for Field Programmable Gate Arrays (FPGAs). The approach attempts to identify alternative wires and alternative functions for wires that cannot be routed due to the limited routing resources in FPGA. The alternative wires (in the logic level) that can be routed through less congested areas substitute the unroutable wires without changing the circuit's functionality. Allowing the logic blocks to have alternative functions also increases the chance of successful routing. A redundancy addition and removal technique is used to identify such alternative wires. Experimental results are presented to demonstrate the usefulness of this approach. For a set of randomly selected benchmark circuits, on the average, 30%-50% of wires have alternative wires. These results indicate that the routing flexibility can be substantially increased by considering these alternative wires. Our prototype system successfully completed routing for two AT&T designs that cannot be handled by an FPGA router alone. The proposed synthesis technique can also be applied to standard cell and gate array designs to reduce the routing area.

1. Shih-Chieh Chang is with Institute of Computer Science and Information Engineering National Chung Cheng University, Chiayi, Taiwan. Kwang-Ting Cheng is with the Electircal and Computer Engineering Department, University of California at Santa Barbara, Santa Barbara, CA 93106, USA. Nam-Sung Woo is with AT&T Bell Lab. Murry Hill. Malgorzata Marek-Sadowska is with the Electircal and Computer Engineering Department, University of California at Santa Barbara, Santa Barbara, CA 93106, USA.

2. Part of this work was done when the first and second authors worked at AT&T Bell Labs. This work was supported in part by the National Science Foundation under Grant MIP 9117328 and in part by AT&T Bell Laboratories and Xilinx through the California MICRO program.