Layout Driven Logic Synthesis for FPGAs

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

*Electrical and Computer Engineering Department, University of California Santa Barbara.

**AT&T Bell Lab. Murry Hill

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 flexibility of routing. The redundancy addition and removal techniques are 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 the 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 INTRODUCTION

Field programmable gate array (FPGA) consists of an array of identical logic blocks and routing resources. In a Table Look Up (TLU) architecture, each logic block can implement any *m*-input Boolean function.

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 [1] [4] [6] maps the circuit onto logic blocks to minimize the number of lookup tables used. Placement and routing then follow. During the physical design process, 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 in the non-congested area. When the routing fails, there are no systematic ways 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 on 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.

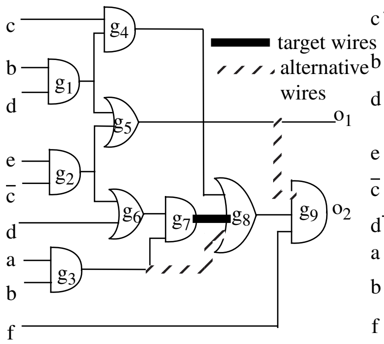


Fig. 1: Example of alternative wires

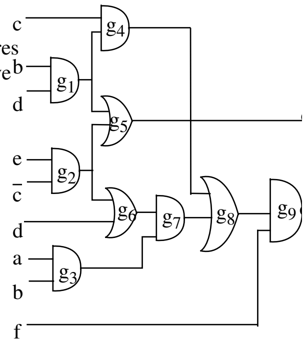


Fig. 2: Example of redundant wire

For example, in the circuit depicted in Figure 1 (from [2]), 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 critical wires. Note that for table-lookup 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 Figure 3. After placement, the routing length of wires can be estimated. If an estimated long wire (target wire *t* in Fig. 3) has a shorter alternative wire in logic domain (wire *a* in Fig. 3), 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 [2]. For another example consider Fig. 4. A circuit shown there, is partitioned and placed on two chips. The interconnection wires between chips typically cause long propagation delay. 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. 5. After partitioning a circuit, suppose the pin constraint of a chip is violated. If an interconnection wire has an alternative wire inside a chip, we may reduce the number of interconnection wires by using its (internal) alternative wires. All the above applications explore the alternatives in 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.

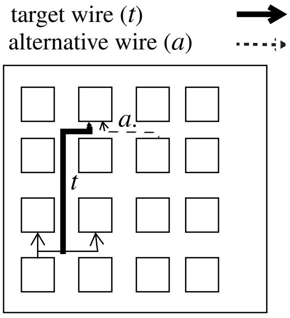


Fig. 3: wire length reduction

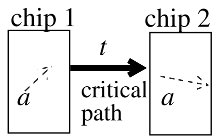


Fig. 4: timing optimization

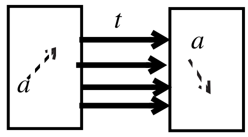


Fig. 5: partitioning improvement

We have developed an efficient method to find alternative wires of a target wire. The method is an extension from the redundancy addition and removal techniques [2][3]. 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 the AT&T ORCA [8] router. 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 reviews the redundancy identification technique that forms a core of our method. Section 3 details the proposed method of finding alternative wires of a target wire. Section 4 discusses the alternative function for a target wire. Section 5 describes the algorithm for the integrated synthesis and routing process. Section 6 shows some experimental results and conclusion follows in Section 7.

2 Redundancy Identification Based on Mandatory Assignments

The alternative wires, when added to the circuit, make the target wire redundant. In a combinational circuit, a wire is redundant if and only if the corresponding stuck-at fault is untestable. We will review the idea in [3] that identifies redundancies using the concept of mandatory assignments.

Definition: The *absolute dominators* (dominators)[5] of a wire W is a set of gates G such that all paths from wire W to any primary output have to pass through all gates in G. For example, the dominators of g_1->g_4 in Fig. 2 are g_4, g_8, *and* g_9.

Definition: A gate is in the *transitive* fanin(fanout) of a wire, if there is a path from the gate to the wire (from the wire to the gate).

Consider the absolute dominators of a wire W. Let *side inputs* of a dominator be its inputs not in the transitive fanout of the wire W. To generate a test for a stuck-at fault at wire W, all side inputs of the wire W's dominators must be assigned their "non-controlling" values. The non-controlling value is 1 for AND(NAND) gate and 0 for OR(NOR) gate. For example, in Fig. 2, to test wire (g_1->g_4) s-a-1, we must assign 1 to c, 0 to g_7, and 1 to f.

When testing a wire stuck-at fault, 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[3] SMA(f) that can be computed via implication [5][7] and recursive learning [8]. If the mandatory assignments of a stuck-at fault test cannot be consistently justified, the fault is untestable and therefore, the wire is redundant.

For example, we like to know whether wire g_5->g_9 can be added in Fig. 2. We need to know whether g_5->g_9 s-a-1 is testable. The mandatory assignment*s* 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.

3 The Alternative Wires of a Target Wire

A wire to be removed is referred to as the target wire. The corresponding stuck-at fault is called the target fault. 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 inconsistent mandatory assignment for the target fault. The idea is originally proposed in [3]. However, the objective in [3] was to minimize the number of connections in a multi-level Boolean circuit while our objective is to remove unroutable wires. Therefore, we allow adding several wires (gates) to replace one unroutable wire. Besides, we include new circuit transformations to remove a target wire.

3.1 Single-Wire Alternative

We say that a wire w_1 is a single-wire alternative of a target wire if w_1 can replace the target wire. This section describes a procedure that finds single-wire alternatives of a target wire.

This procedure consists of three steps. In the first step, we calculate the SMA of the target wire stuck-at-fault. Then, we identify a set of candidate connection wires to be added. Each candidate connection when added to the circuit will cause the SMA to become inconsistent and thus make the target wire redundant. However, adding such a candidate connection may change the circuit's behavior. Therefore, we need to check whether the 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. The following example illustrates the process of finding single-wire alternatives.

Example: Consider the circuit in Fig. 2. 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, f=1}. Note that g_5 is outside the transitive fanins and 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 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 and therefore, wire g_5->g_9 is an alternative wire for wire g_1 ->g_4.

Given a target wire, Fig. 6 shows four different types of candidate connections to be added to make the target wire redundant. The doted 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 the wire w.

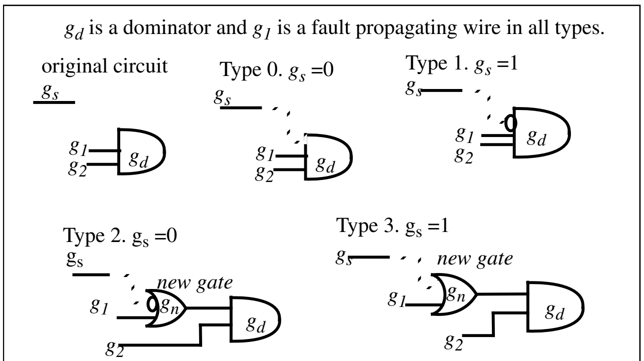


Fig. 6: Types of transformations

All candidate connections (g_s g_d), in Fig. 6, use the same principle described as follows. The source gate g_s has a mandatory assignment. The destination gate g_d is a dominator. After making any transformation shown in Fig 6., the SMA of the target wire becomes inconsistent. Type 0 and Type 1 in Fig. 6, add a wire from g_s to the dominator g_d. Type 2 and Type 3 add a new gate g_n to which all fault propagating wires previously connected to g_d are reconnected to g_n and connect a wire from g_s to g_n. For example, in Type 2, let g_1 be 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 a non-controlling value 1 which causes a conflict with the original mandatory value 0. We can also derive similar transformations when the dominator is an OR gate.

Fig. 7 shows the 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_array.
    if (g_i is a dominator)  insert g_i into the destination_array.
  }
  for (each g_s in the source_array) {
    for (each g_d in the destination_array) {
      add_wire_gate(g_s, g_d, type_0or1);
      if(verify_redundant(g_s, g_d, type_0or 1)) return (g_s, g_d, type_0or1);
      rm_wire_gate(g_s, g_d, type_0or1);
      add_wire_gate(g_s, g_d, type_2or3);

      if(verify_redundant(g_s, g_d, type_2or3)) return(g_s, g_d, type_2or3);
      rm_wire_gate(g_s, g_d, type_2or3);
    }
  }
}
```
Fig. 7: Pseudo code for finding single-wire alternative

3.2 Multiple-Wire Alternatives

If none of the candidate connections is redundant, there are no single-wire alternatives for the target wire. In this section, we describe a method 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. Suppose there are 6 candidate connections, w_0,..., w_5, suggested by the procedure of Fig. 7 (searching for single-wire alternatives) and no candidate connection is redundant. Each candidate connection wire is then 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 adding w_3. 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 as follows.

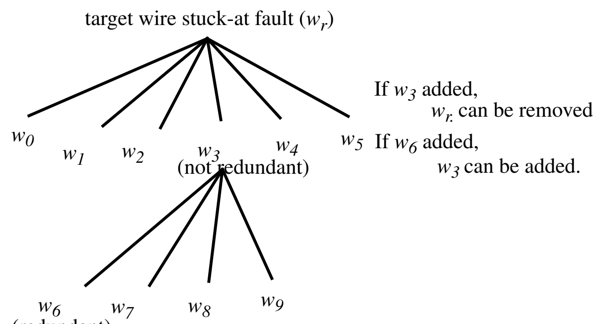


Fig. 8: Identifying multiple-wire alternatives

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 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 Using alternative functions of look-up tables to remove a target wire.

We now show how to apply the alternative wire technique for lookup table architecture based FPGAs. One way is to decompose a mapped FPGA into primitive gates and use the technique described in the previous section to replace unroutable wires. However, for lookup table based FPGAs, the technique can be further generalized by considering *alternative functions* of an unroutable wire. The generalized technique directly operates on the lookup table level without decomposing the mapped LUT into primitive gates.

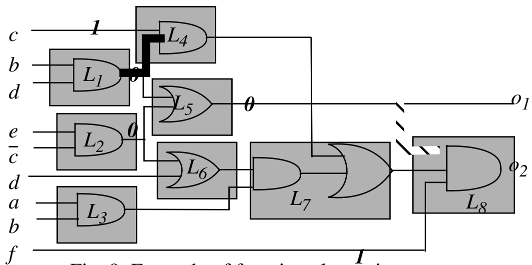


Fig. 9: Example of function alternative

4.1 Alternative functions of lookup tables

A k-input lookup table 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 are the inputs of 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.

Similarly to the procedure of finding alternative wires, the process of finding alternative functions consists of three steps. In the first step, we calculate the SMA of the target wire stuck-at fault. In the second step, 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. 9 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. In the second step, we find $f'_{L8}(L_5, L_7, f) = L_5 \bullet L_7 \bullet f$ (the original function is $f_{L8}(L_5, L_7, f) = f \bullet L_7$) 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, 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.

In the second step of finding candidate functions, we only consider absolute dominators of the target wire. The function of the dominator's lookup table will be considered for changing if (1) the dominator has any unused input, or if (2) the dominator has no unused inputs but it has an input with a mandatory assignment. When the dominator has any unused input, we connect to it a wire, an extra-input, which has a mandatory assignment. In the example of Fig. 9, L_8 is one of the dominators and the extra-input L_5 has a mandatory value 0. The procedure for deriving the candidate functions is based upon the following procedure.

After calculating the SMA of the target wire, suppose the LUT L_i is a dominator and a wire w_x is one of L_i's inputs which has a mandatory value v. Among those inputs to L_i, some inputs denoted X_1 are in the transitive fanout of the target wire. Note that w_x is an input variable in L_i. We define a candidate function f'_{Li} by specifying the output of each input vector as follows. Given an input vector, when the w_x variable is $!v$, we set $f'_{Li} = f_{Li}$ and when w_x is v, f'_{Li} is chosen to be any function independent of X_1. If there is no input wire of L_i with mandatory assignment, the new function f'_{Li} will be created by connecting a wire with mandatory assignment to an unused input.

Lemma: If we replace f_{Li} with f'_{Li}, a candidate function from the above procedure, the SMA of the target wire stuck-at-fault is inconsistent.

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. 10: The inconsistency between old function and candidate function must be don't cares.

Continuing the example from Fig. 9, Fig. 10 illustrates the lemma. Suppose L_8 implements an AND function in Fig 10a and for the given target fault, L_5 has a mandatory value 0. We assume the target wire is in the transitive fanin of L_7. There are 4 candidate functions as shown in Fig 10(b,c,d,e). Each of the four functions is the same as the original function *($L_7 \bullet f$)* when L_5 =1 and is independent of L_7 (which is in the fault propagating path) when L_5 =0.

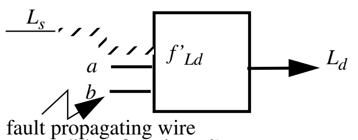
After obtaining these 4 candidate Boolean functions, we need to check whether they are valid. First, we compare the old function with each of candidate functions. For example, in Fig. 10, the discrepancy between the original function (a) and function (b) occurs in the minterm (L_5, L_7, f) =(0, 1, 1) 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. $f'_{L8}(0, 1, 1)$ can be assigned to either 0 or 1 and the function of the circuit does not change. We show how to check whether some minterm is a don't care in the following procedure.

Suppose we try to verify whether (v_1, v_2,...,v_n) is a don't care cube in a function $f_{Lm}(L_1, L_2, ..., L_n)$. We set the mandatory assignment of L_i to the corresponding v_i. Considering the example in Fig. 9, to verify cube (L_5, L_7, f) = (0, 0, 1) in L_8 is a don't care, we set L_5=0, L_7=0, f=1 to be their mandatory assignments. Then, we calculate the SMA as described in section 2.

Lemma: If the SMA cannot be consistently justified by the above procedure, the cube (l_1, l_2,...,l_n) in $f_{Lm}(L_1, L_2, ..., L_n)$ is a don't care in the circuit.

The overall algorithm for finding an alternative function of a target wire is summarized in Fig.11.

Step 1: Calculate the SMA of target wire stuck-at fault.
Let L_s have a mandatory assignment v(0 or 1) and L_d be a dominator.



Step 2: Suggest candidate functions f'_{Ld}:

$fun'_{Ld}(L_s != v) = f_{Ld}$

$fun'_{Ld}(L_s = v)$ = a function that is independent of fault propagating inputs

Step 3: Test if any candidate function is valid:
If all cubes that belong to $f'_{Ld} \oplus f_{Ld}$ are don't cares, the candidate function is valid.

Fig. 11: An algorithm to find alternative functions of a target wire

4.2 Decomposition into gates

To apply the approach described in Section 3 to lookup table architecture, we can simply 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 lookup tables. An *internal* wire is inside a 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.

Let each alternative wire be expressed as (w_s, w_d) where each of w_s and w_d represents a gate. We assign a cost to an alternative wire based on the following rules. These rules try to calculate the number of external wires added on the circuit (Fig. 12 illustrates these rules):

- Rule 1: cost =0, if w_s and w_d are both inside the same lookup table.
- Rule 2: cost =1, if the lookup table of w_d has less than k inputs and w_s is an output of another lookup table.
- Rule 3: cost = p, if the lookup table of w_d has less than k inputs and w_s is inside a lookup table. Since w_s is not the output of a lookup table, we need to create a new *(p-1)* input lookup table to duplicate the function that creates w_s.
- Rule 4: cost = infinite, if the lookup table containing w_d has k inputs and w_s is in another lookup table. Note that we may use Shannon decomposition to create 3 new 5-input lookup tables but we choose not to consider this substitution[6].

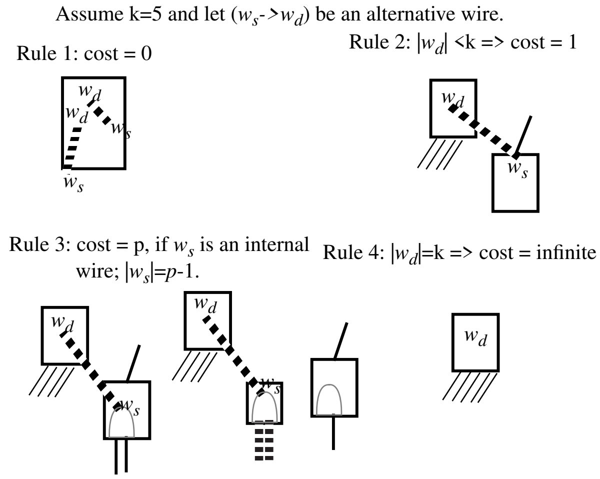


Fig. 12: Costs of alternative wires

5 Integration of routing and synthesis

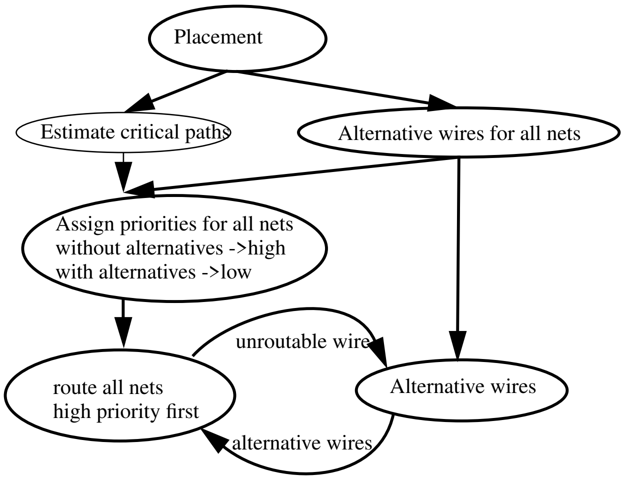


Fig. 13: The flow of layout driven synthesis

In previous sections, we have presented a synthesis technique to find the alternative wires of a target wire. To fully utilize the alternative wire technique, we precompute the wire alternative information and pass it to the routing tool before routing starts. According to the wire alternative information and timing information (obtained from other tools), a router sets a priority for each wire. Then, it routes the wires in such an order that wires without alternatives and/or in timing critical paths are routed 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 can not be routed, we also try ripping up some wires around the congested area. Figure 13 shows the flow of the layout driven synthesis process.

6 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 [6]. For industry examples, we used AT&T ORCA [8] technology mapper. Then, we decomposed each lookup table into AND, OR gate circuit as described in Section 4.2. 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 a 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 forth column shows the number of wires that have triple-wire alternatives and the number in parenthesis shows the total number of triple-wire alternatives. The cost of an alternative wire is defined in Section 4.2. The fifth and sixth columns show CPU time running on a SPARC 10.

In our second experiment, we linked our synthesis technique with AT&T ORCA router. Two unroutable circuits were studied. The first circuit had 169 5-input lookup tables and 738 external wires. Among these external wires, we found that 121 wires had single-wire alternatives and 201 wires had triple-wire alternatives. If we run the router using the timing-driven option, the router failed to connect 4 wires. We applied our technique to this circuit and successfully completed the routing. The second circuit had 495 5-input lookup tables and 1980 external wires. Totally 1115 wires had single-wire alternatives and 2001 wires had triple-wire alternatives. There was one unroutable wire in the original design. By replacing the wire by its single-wire alternative, the routing was successfully completed.

7 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.

Acknowledgment. This work was supported in part by the National Science Foundation under Grant MIP 9117328 and in part by AT&T Bell Laboratories and Digital Equipment Corporation through the California MICRO program.

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
mis2	122	53(84)	77(305)	43.0	218.9
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
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

8 Reference

[1] Shih-Chieh. Chang and Malgorzata Marek-Sadowska, “Technology Mapping via Transformations of Function Graphs”, *Proc. IEEE International Conference on Computer Design*, pp. 159-162, 1992.

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

[3] Luis 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.

[4] 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.

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

[6] 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.

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

[8] “ORCA: A NEW Architecture for HIgh-Performance FPGAs”, *AT&T technical report*.

[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.