Synthesis for Multiple Input Wires Replacement of a Gate for Wiring Consideration

Shih-Chieh Chang, Jung-Cheng Chuang, Zhong-Zhen Wu
National Chung-Cheng University, Jay-Yi, Taiwan R.O.C.

Abstract

The alternative wire technique attempts to replace a target wire by another wire without changing the logic functionality. In this paper, we propose two new transformations of replacing wires. One transformation simultaneously replaces multiple input wires of a gate by a new set of input wires and the other performs gate decompostion during the alternative wire process. To accomplish such complex transformations, we discuss some theoretical foundations for replacing multiple wires. Understanding how wires/gates can be replaced by other wires/gates allows us to speedup the process tremendously.

1 Introduction

In the traditional design flow, interconnect design takes a (fixed) net list and attempts to optimize the interconnects from the given net list which can be produced from a logic synthesis tool. To handle the growing complexity of circuit design, designers are forced to look ahead in a higher level of abstraction and propose algorithms which combine logic synthesis and layout synthesis. There have been several research papers [2][4][5][7][9][11] integrating the logic and layout synthesis algorithms. The work [9] proposes an earlier synthesis strategy which uses "wireplanning" to distribute delays over the functional elements and interconnects. In [11], they present an integrated design flow which combines floorplanning, technology mapping, and placement using a dynamic programming algorithm.

There are also several logic synthesis techniques [1] [2] [3] [6] [12] which may potentially be modified to consider the wiring effect. Among those, [2] proposes a post-layout logic synthesis technique which allows one to modify wire topology. This technique, called the alternative wire technique (Alwire) explores the freedom in the logic domain to provide additional flexibility in the layout area. The basic philosophy of Alwire is to remove some wire by adding another wire to the circuit. For example in Fig. 1 by adding wire g_1->g_5, wire c->g_2 can be removed without changing the logic functionality.

The traditional techniques, though, can be quite effective in performing single wire replacement. Due to the large searching space required, to the best of our knowledge, there is very little success in simultaneously replacing multiple wires/gates. In this paper, we propose two new transformations. For the (direct) input wires of a node, one transformation simultaneously replaces the old set of input wires by a new set of input wires, assuming that the old set of input wires is targeted for replacement. For example, in Fig. 2, suppose the input wires (e->g, f->g) of node g, are under consideration for removal. Our transformation can identify that a new set of wires *(o->g, p->g, q->g)* can replace the old set (*e->g, f->g*) without changing the circuit functionality. The other transformation proposed in this paper performs gate decomposition during the alternative wire process. In Fig. 5, only after decomposing the big OR gate g_1 *(*Fig. 5a) into two small OR gates, g_6 and g_7 (Fig. 5b), we can then replace wire c->g_2 by wire g_6->g_5.

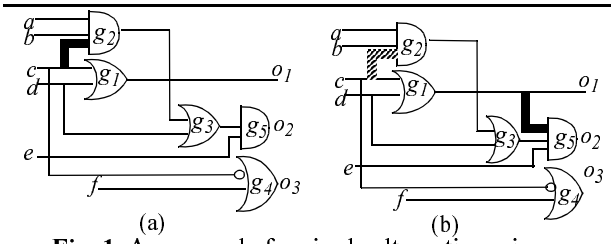

Fig. 1 An example for single alternative wire.

We also would like to mention that there are several logic optimization algorithms adding multiple wires/gates and then removing wires/gates to reduce the size of a circuit. However, those algorithms do not have specific target wires for removal in mind and, therefore, are quite different from the objective of this paper.

In order to achieve such complex transformations, we discuss some theoretical foundations for replacing multiple wires. Understanding how wires/gates can be replaced by other wires/gates allows us to speedup the process tremendously. As a result, we can perform these complex transformations which were unfeasible in practice in [2]. Our experimental results show that these new transformations require reasonable run time but can provide much more flexibility than those in [2].

2 The Alternative Wire Technique

In this section, we briefly review the Alwire technique. Basically, the Alwire technique 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-

0-7803-5832-X /99/$10.00 ©1999 IEEE.

at fault undetectable (redundant) and therefore it can be removed. Finally, the technique checks whether a 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. 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 $\{a=1, b=1, d=0, e=1, c=0, g_1=0, o_3=1\}$ must be set. These assignments are called the *mandatory assignments (MAs)* for $c->g_2$ s-a-1 test. If the (non-existing) wire $g_1->g_5$ is added to the circuit, the MA of $g_1=0$ will cause node g_5 to have the MA of 0 which blocks the fault propagation and cause 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 $c->g_2$.

In the Alwire procedure, if there are k candidate connections, it requires k times of redundancy checking to find all possible alternative wires. In the case of simultaneously replacing l wires which all have k candidates, the complexity will be $O(k^l)$. In a large circuit, the value k can be very large. Therefore, directly applying the Alwire technique for multiple wire replacement is very costly.

3 Gate support replacement

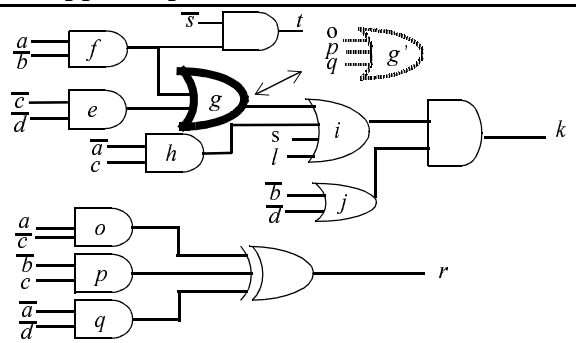

Fig. 2 Replace inputs of g by new set of inputs (o, p, q).

In this section, we discuss a transformation which replaces multiple wires at the same time. The transformation is termed the *alternative node* technique (*Alnode*) in which we attempt to replace the input wires of a node by a new set of input wires. The size of a new set can be larger or smaller than the original set. For example, in Fig. 2, the inputs of node g, (e, f) can be replaced by the new set of inputs (o, p, q). In this case, after replacement, nodes e can be removed from the circuit. Sometimes, *Alnode* may lead to area reduction or good routing results.

Before we discuss the procedure for *Alnode*, we would like to distinguish different types of MAs. We define the *observability MAs* of wire w, *obvMAs(w)* to be MAs assigned only for propagating a fault from wire w to a Primary Output (PO). For example, in Fig. 2, to propagate a fault from wire $g->i$ to a PO, the observability MAs $obvMAs(g->i)=\{h=0, s=0, l=0, j=1\}$ must be set. Similarly, we can define the observability MAs of node n, *obvMAs(n)*, so that a fault can be propagated from n to a 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 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 is equal to $obvMAs(n->m) \cup actDrvMAs(n=0)$. For example in Fig. 2, the MAs for wire $g->i$ s-a-1 test is equal to $obvMAs(g->i) \cup actDrvMAs(g=0)$ where $obvMAs(g->i) = \{h=0, s=0, l=0, j=1\}$. The set $actDrvMAs(g=0)$ can be determined from the following: Under the observability conditions of $\{h=0, s=0, l=0, j=1\}$, we assign MA $g=0$ which can lead to MAs $\{e=0, f=0\}$. We then show MA $e=0$ can lead to MA $q=0$ in Fig. 3.

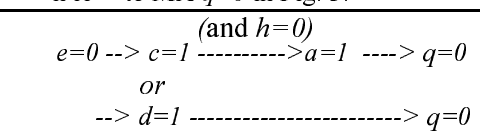

Fig. 3 MA $e=0$ leads to MA $q=0$.

Due to space limitation, we do not show other derivations of MAs. However, using the similar technique (recursive learning [8]), one can derive $\{o=0, p=0, q=0, t=0\}$ from $\{e=0, f=0\}$. As a result, we have $actDrvMAs(g=0) = \{g=0, e=0, f=0, o=0, p=0, q=0, t=0\}$. The MAs in $actDrvMAs(g=0)$ are MAs implied from $g=0$ under the conditions of observability MAs. Intuitively, one can consider that a node in $actDrvMAs(g=0)$ has some "relation" with node g.

Now, we present our *Alnode* algorithm. This algorithm attempts to replace node n by a new node n' whose support set is different from those of node n. Node n' is called an *alternate node* for n. We also assume that **none** of the input wires of node n and n' are redundant. For ease of discussion, let us assume that node n is a 2-input OR gate with input nodes $\{u, v\}$ and its alternative node n' is a 3-input OR gate with input nodes $\{x, y, z\}$. Therefore, our objective is to select three nodes from existing nodes in a circuit to replace the old set $\{u, v\}$. For the case of AND gates or other size of inputs, this can be easily derived. In order to replace node n by a new node n', the following lemma must be satisfied.

Lemma 1: Suppose the observability MAs *obvMAs(n)* are set first. If node n' can replace node n, then,

$$\text{MA } n=0 \text{ must imply MA } n'=0, \text{ and} \qquad \text{(EQ 1)}$$

$$\text{MA } n=1 \text{ must imply MA } n'=1 \qquad \text{(EQ 2)}$$

Lemma 1 states that, if a Primary Input (PI) vector evaluates a 0 (1) at node n, the vector must also evaluate a 0 (1) at an alternative node n' under the condition that the vector can propagate a fault from n to a PO. Therefore, to check whether node n' with a new set of inputs can replace the old node n, we only need to check whether the above equations EQ 1 and EQ 2 are satisfied. According to EQ 1, node n' must have an MA of 0 so the equation EQ 1 can be re-written to EQ 3 and so does EQ 2 to EQ 4.

$$\text{MA } n'=0 \in obvMAs(n) \cup actDrvMAs(n=0) \qquad \text{(EQ 3)}$$

$$MA\ n'=1 \in obvMAs(n) \cup actDrvMAs(n=1) \quad \text{(EQ 4)}$$

Let us define the union sets of MAs in EQ 3 and EQ 4 to be *UMAs_0* and *UMAs_1*:

$$UMAs\_0 = obvMAs(n) \cup actDrvMAs(n=0) \quad \text{(EQ 5)}$$

$$UMAs\_1 = obvMAs(n) \cup actDrvMAs(n=1) \quad \text{(EQ 6)}$$

Due to the assumption of node *n'* being an OR gate, to satisfy *n'=0* in EQ 3, all three inputs (*x, y, z*) of *n'* must also have MAs of *0* in UMAs_0. Therefore, to select (*x, y, z*), we only need to search for nodes in UMAs_0. A simple algorithm to find an alternative node with three inputs is as follows. Suppose (*x, y, z*) are such nodes which all have MAs of 0 in UMAs_0. We can then form a new OR gate *n'* by connecting nodes *x, y* and *z* as its inputs. Since nodes *(x, y, z)* are from UMAs_0, node *n'* can satisfy the condition of EQ 3 (or EQ 1). The subset (*x, y, z*) is called a ***candidate*** *size-3 subset* for *n*. For a candidate subset to form an alternative node, still, the candidate node need to satisfy the condition of EQ 2.

For example, in Fig. 2, suppose we would like to replace the input wires of node *g*. First, we compute UMAs_0 in EQ 5. Because of *obvMAs(g) = obvMAs(g->i) = {h=0, s=0, l=0, j=1}* and *actDrvMAs(g=0) ={g=0, e=0, f=0, o=0, p=0, q=0, t=0}*, we have UMAs_0 = *{h=0, s=0, l=0, j=1, g=0, e=0, f=0, o=0, p=0, q=0, t=0}*. There are 11 MAs in UMAs_0 so the total number of size-3 candidate subsets is C_3^{11} =165. We can now form a new OR gate *g'* with inputs *o*, *p* and *q* shown as the dotted gate in Fig. 2. Then, for a candidate subset to form an alternative node, we still need to verify the condition of EQ 2, i.e., to check if *g=1* can imply *g'=1*. With some computation (described later), one can find that EQ 2 can in fact be satisfied so node *g'* with inputs *o, p*, and *q* is an alternative node of *g*.

Since node *g* is an OR gate, by EQ 2, it is difficult to imply other MAs from MA *g=1*. Instead, we can rewrite EQ 2 to the following equivalent equation: Under the conditions of observability MAs,

MA *n'=0* must imply MA *n=0* (EQ 7)

All three equations EQ 2, EQ 4 and EQ 7 are equivalent. However, when node *n* and *n'* are OR gates, implications of MA of *n'*=0 by EQ 7 can be easier than the implications of *n=1* by EQ 2. For the same example, let us check whether the candidate subset (*o, p, q*) can satisfy the condition of EQ 7. By EQ 7, we check whether *g'=0* implies *g=0*. The derivation is shown in Fig. 4.

g'=0 -->o=p=q=0
o=0=> *-->a=0* (and *q=0*) *----------->d=1, f=0-->e=0-> g=0*
or
-->c=1----------->b=1,e=0-->f=0-->g=0 (and *p=0*)

Fig. 4 MA *g'=0* implies MA *g=0*.

In summary, to find all possible alternative nodes (with k inputs) for node *n*, the algorithm first generates all possible candidate size-k subsets from UMAs_0. Then, for each candidate subset, the algorithm checks whether the condition of EQ 7 is satisfied assuming the inputs of node *n'* are nodes in the candidate subset. Since the MA set, UMAs_0 can be large in a circuit, to enumerate all possible candidate subsets may be very time-consuming. In the following, we discuss a way to reduce this complexity. Again, assume that nodes *n* and *n'* are OR gates.

Theorem 2: If node *n'* is an alternative node of node *n,* an input of node *n'* must NOT belong to *obvMAs(n)*.

Let node *z* be an input of node *n'*. According to Theorem 2, node *z* must not belong to *obvMAs(n)*. Since node z must be selected from *UMAs_0=obvMAs(n) ∪ actDrvMAs(n=0)*, node *z* belongs to nodes in *actDrvMAs(n=0)*. For the same example, to form a candidate subset, instead of selecting nodes from UMAS_0=*{h=0, s=0, l=0, j=1, g=0, e=0, f=0, o=0, p=0, q=0, t=0}*, we only choose nodes from *actDrvMAs(g=0) = {g=0, e=0, f=0, o=0, p=0, q=0, t=0}*.

Again, we assume that the target node *n* is an OR gate with inputs *{u, v}* and wires *{u->n, v->n}* are not redundant. Let us consider *u->n s-a-1* fault. After computing *obvMAs(u->n)*, we can obtain *actDrvMAs(u=0)*. Similarly, we can obtain *actDrvMAs(v=0)*. So, we have the following theorem.

Theorem 3: Among the input nodes of *n'*, there is at least one input node which has an MA in *actDrvMAs(u=0)* and similarly there is at least one input node which has an MA in *actDrvMAs(v=0)*.

From Theorem 3, a candidate subset must contain at least one node in *actDrvMAs(u=0)* and at least one in *actDrvMAs(v=0)*. After applying Theorem 2 and Theorem 3, we can reduce from 165 to 30 candidate subsets. Among 30 candidate subsets, only *{(e, o, p) (f, o, q), (o, p, q)}* can pass the verification of EQ 7 so each of these subsets can form an alternative node.

4 Gate decomposition Alternative wire.

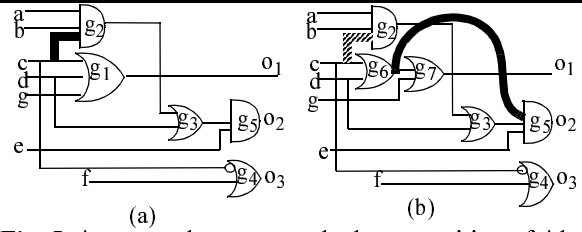

Fig. 5 An example source node decomposition of Alwire.

In the original Alwire technique, the source (destination) node of an alternative wire can only be selected from an existing node in a circuit. In this section, we extend the ability of Alwire technique to consider the gate decomposition.

The gate decomposition technique contains two different types, source and destination decomposition. Without going into details, we will use an example to describe the source decomposition tech-

nique. The destination decomposition can be easily extended. Consider the circuit in Fig. 5a which is a modified circuit of Fig. 1a by adding one additional input, g to g_1. Without decomposition, we cannot find any alternative wire for c->g_2. However, if we decompose the big OR gate of g_1 into two small OR gates, g_6 and g_7 in Fig. 5b, we can then pull out the function of $(c+d)$. In this way, we can find an alternative wire g_6->g_5 for c->g_2 which is similar to the method in Fig. 1. Consider to replace a wire x->y where node y is an AND gate. In our algorithm, an AND/OR node n is considered for decomposition when it satisfies the following conditions: 1. node n does not have an MA but several of its inputs have MAs and 2. one of its inputs has an MA in $actDrvMAs(x=0)$. These conditions allow us to efficiently achieve the Alwire technique with the source node decomposition.

TABLE 1. The results of Alnode.

circuit	# gates	alnodes k inputs	run time (sec)	alnodes k+1 input	cpu (sec)
apex6	562	3572	112.54	58553	1468.21
apex7	182	3820	126.18	48756	5770.68
b9	102	389	33.58	1004	130.27
C2670	694	4200	19.22	30018	243.44
C432	175	5356	15.69	43694	425.58
C499	370	64	31.41	1869	287.81
C6288	2353	1544	11.8	8170	136.7
C880	302	181	32.75	1029	100.66
cht	184	251	11.43	1105	29.7
cmb	29	545	4.9	1011	10.1
comp	120	49	1.34	163	2.79
lal	105	14532	199.94	72520	2574.38
mux	76	36	0.31	90	0.64
my_adder	177	224	0.34	365	0.89
rot	453	3573	23.29	17133	143.3
sct	81	6981	303.28	48482	2838.2
total 33	1	7.91		54.95	

5 Experimental results.

We have implemented the two transformations mentioned in Section 3 and Section 4 and performed experiments on a set of MCNC and ISCAS benchmarks. The circuits are first optimized by SIS (algebraic) script and then decomposed into AND/OR gates with maximum of five inputs. (Without this restriction, some circuits such as f51m have too many alternative nodes.) The first experiment attempts to find all possible alternative nodes for each node in a circuit in Table 1. Column one shows the name of the circuit and Column two shows the number of gates in the circuit. In the third and fourth column, we show the results in which an alternative node has at most the same number of inputs as those of the replaced node. Column three shows the summation of such alternative nodes in a circuit and Column four shows the cpu run time. In the fifth and sixth column, we show the results in which an alternative node has at most the same number of inputs as (the number of inputs of the replaced node)+1. Column five shows the total alternative nodes and Column six shows the cpu run time. The last row in Table 1 shows the average alternative nodes of a node. In Table 2, we show the results of alternative wires allowing gate decomposition. Column two shows the total number of wires in a circuit. The third and fourth columns show the number of alternative wires and the corresponding run time. The fifth and sixth columns show the number of alternative wires with decomposition and the corresponding cpu time. These results show that with new transformations, we can obtain more flexibility than the Alwire technique. Calculations were performed on a Ultra 1.

6 Conclusion

In this paper, we have proposed two new complex transformations. Our experimental results show that with reasonable run time, these two transformations can increase flexibility more than those in the alternative wire technique for wiring consideration.

7 References

[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] S. C. Chang, K. T. Cheng, N.S. Woo and M. Marek-Sadowska "Layout Driven Logic Synthesis for FPGA," *Proc. Design Automation Conf.* pp. 308-313, June 1994

[3] 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.

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

[5] W. Gosti, A. Narayan, R. Brayton, A. L. Sangiovanni-Vincentelli, "Wireplanning in Logic Synthesis," *Proc. ICCAD, pp. 26-33, 1998.*

[6] 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.

[7] K.Keutzer, A.R. Newton, and N. Shenoy, "The future of logic synthesis and Design Planning," Proc. IWLS, 1998.

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

[9] R. H.J. Otten, R. K. Brayton, "Planning For Performance," Proc. DAC, pp. 122- 127, 1998.

[10] M. Pedram and N. Bhat, "Layout Driven technology Mapping," *Proc. DAC, pp. 99-105, 1991.*

[11] A. H. Salek, J.Lou, M. Pedram, "A DSM Design Flow: Putting Floorplanning, Technology Mapping, and Gate-Placement Together," Proc. DAC, pp. 128-133..

[12] M. Yuguchi, Y. Nakamura, K. Wakabayashi, T. Fujita "Multi-Level Minimization based on Multi-Signal Implications", proc. DAC, 1995, pp. 658-662.

TABLE 2. The results of Alwire with decompostion.

circuit	# wires	# alwires	cpu (sec)	# decomp alwires	cpu (sec)
apex6	1228	1387	404.38	1511	488.79
apex7	414	1285	62.88	1380	65.93
b9	221	735	23.7	791	23.8
C2670	1570	3897	1557.26	4943	1616.67
C432	387	1071	129.98	1074	131.7
C499	784	64	193.33	64	192.79
C6288	4705	1419	149.75	1419	153.94
C880	648	823	93.19	832	94.65
cht	384	360	25.58	477	27.32
cmb	77	465	15.71	472	15.98
comp	250	886	88.71	886	88.5
lal	258	3633	61.38	3959	64.27
mux	162	995	16.16	999	16.73
my_adder	337	561	8.69	561	9.14
rot	1052	1805	188.94	2072	202.37
sct	207	2084	65.94	2245	69.9
total 33	1	1.76		1.93	