Circuit Partitioning with Coupled Logic Restructuring Techniques

Yu-Liang Wu
Dept. Computer Sci. & Eng.
The Chinese University of HK
Shatin, NT, Hong Kong

Xiao-Long Yuan
Dept. Computer Sci. & Eng.
The Northwest Polytechnical University
Xi'an, 710072, China

David Ihsin Cheng
Ultima Interconnect Technology
Sunnyvale, CA 94089
USA

Abstract - Traditionally, the circuit partitioning is done by modeling the circuit as a graph and the partitioning is carried out without altering the circuit itself. Applying the technique of circuit rewiring, the partitioning can be further improved by doing some local logic perturbation along the cut-line to drag the solution out of some local minimal. In this paper, we propose an effective coupling scheme of two powerful rewiring techniques to further improve upon those already selected best partition results produced by other conventional partition tools. The improvement is attributed to the additional capability of exercising a guided circuit perturbation, which was not available in the conventional schemes.

The known ATPG-based and the recently proposed graph-based rewiring techniques compose our coupling scheme. The ATPG-based rewiring technique is very flexible; however the graph-based rewiring technique is faster and can couple quite well with the ATPG-based scheme to exploit a much larger room for logic perturbations. Our encouraging experimental results show that these two techniques couple each other quite well for this application without costing much CPU overhead. This scheme is also quite efficient thus should be very useful for large circuits.

1. Introduction

Circuit partitioning is the task of dividing a given circuit into some smaller sub-circuits. As the design scale increases, this task plays a very important role in circuit design automation. For example, it has been shown that because of the limitation on the number of pins, which corresponds to the number of cut nets, the utilization of chip area in multiple chip emulation system is limited to only 10%-20% [1].

Traditionally, the task of circuit partitioning is performed by first modeling the circuit as a graph (or hypergraph), then partitioning is applied on this graph. It is known that the graph partitioning problem is NP-hard [2]. The commonly used graph partitioning algorithms can roughly be classified into three categories. The first category strictly abides by the modeled graph, with the graph unchanged. The second category of algorithms [5][1] can change the graph through node replications. Major improvements on the partitioning are obtained by sacrificing some area. All of these two categories of algorithms perform the partitioning task on the graph without considering the function of the circuit. The third category of partitioning algorithms [6][7][8][9] couples the graph information (nodes and their connections) and circuit function (node's function). We find that although the algorithms in [6] and [7] can improve the partitioning results, their computation costs are expensive. [8] can only be applied to FPGA circuits.

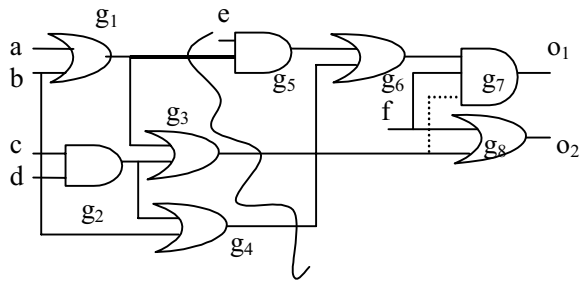
As adding wires (single or multiple redundant wires) and gates to a circuit may lead to the elimination of more wires and gates on the circuit without changing the circuit functionality, these so-called alternative wiring (rewiring) techniques have been widely used to attack many logic level and physical level design problems and have obtained quite good result [10] [11] [12] [9] [13] [14] [15]. RAMBO [10-15] is known to be a very effective technique of this kind. Replacing a wire on the critical path by a shorter alternative wire elsewhere may cut shorter the critical path and produce a faster circuit. Substituting an unroutable wire within a congested area with a routable alternative wire in a less congested area can help improve the task of routing. Similarly, the techniques can be used to remove wires located on the partition cut line and thus reduce the cut size of the partition. [9] is the only existed algorithm so far that uses the alternative wire technique (ATPG-based) to improve the partitioning results. It has a relative small computation cost in comparison with other partitioning algorithms. It combines the alternative wire technique with a partitioning algorithm to obtain a better partitioning result. Because the partitioning algorithm in [9] always select a non-negative-gain wire to replace the target wire; thus, it may yield to a local optimal solution. Also, it only considers the single case of adding one wire to replace another wire. In fact, there exist other cases of replacing a target wire, such as adding a gate and a wire to replace another wire, adding two gates and a wire to substitute a wire, etc. Therefore, some possible solution domain may be omitted in [9].

To investigate the possibility of perturbing the circuit without applying any Boolean operations, we study the minimal circuit structure patterns that can yield alternative rewiring transformations. We observe that some such minimal structures do exist and commonly appear in most practical circuits; therefore it may not be necessary to repeat the ATPG based logic implications to the same pattern each time it is encountered. We also observe that the nearest alternative wire usually locates not too far away from the target wire. Based on these results, a Graph-Based Alternative Wire (GBAW) transformation technique for identifying the alternative wire of a target wire was developed [16] and used in this paper. The philosophy of this technique is to locate matching between a sub-circuit and some "pre-specified" patterns, which includes a target wire and its alternative wire. When a part of the circuit is matched to the pre-specified pattern, the rewiring can be done without performing any logic implication. The major advantage of this technique is the run time. Since the technique does not need to process redundancy test or logic implications, it is very effective in

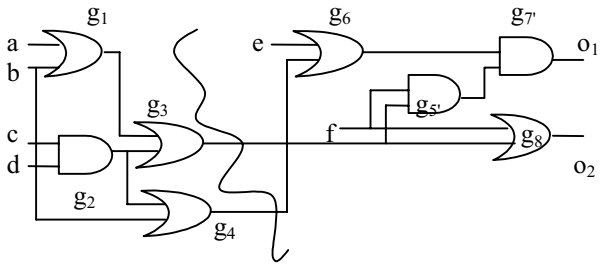
terms of speed. To reduce the search space, we only consider small "pre-specified " patterns in locating close alternative wires. GBAW can also locate a distant alternative wire by the repeated waveform propagation of these local pattern matching.

As GBAW is much faster in locating most local alternative wires (due to the elimination of logic implication), while RAMBO is more flexible in locating some alternative wires, in this paper, an algorithm RG (RAMBO and GBAW), which couples these two techniques for circuit repartitioning is proposed.

In our proposed approach, we first apply the well known Fiduccia-Mattheyses (FM) partitioning algorithm [17] to partition a circuit many times and pick the best result for further improvement. We then apply RG to further improve the previously partitioned result through logic perturbation. For example, in Fig. 1(a), there are 3 nets on the cut-line, which is shown by a wave line. However, after substituting the thick line by the dotted line, the cut size is reduced to only 2, as shown in the Fig. 1(b). We conducted some repartitioning experiments on the MCNC benchmarks with RG, the experimental results are very encouraging.



(a) Partition before applying alternative wiring



(b) Partition after applying alternative wiring

Fig. 1 Circuit partitioning before /after rewiring

2 Background and Definitions

A Boolean network is a directed acyclic graph (DAG) where each node g_i is associated with a Boolean function f_i and a Boolean variable y_i.

A wire w_r is an alternative wire of wire w_t, if replacing wire w_t by wire w_r will still keep the circuit functionality unchanged. In this context, we term w_t as the target wire, and wire w_r its alternative wire.

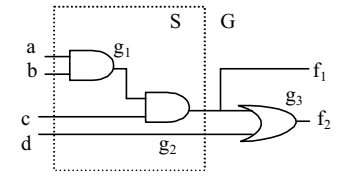
In a Boolean network, the in-degree of a node g, denoted by $d^-(g)$, is defined as the number of edges entering g. The out-degree of a node g, denoted by $d^+(g)$, is defined as the number of edges leaving g.

In this work, a node g is denoted as a triple (op, $d^-(g)$, $d^+(g)$), where op is the Boolean operator of g.

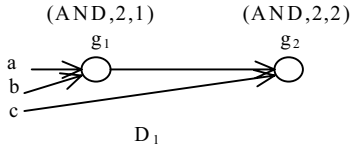
A wire is replaceable if it has an alternative wire.

The Graph-Based Alternative Wire (GBAW) scheme is an alternative wire locating technique, which searches alternative wires by checking graph matching between local sub-networks and some pre-specified minimal sub-graph configurations containing alternative wires within a given distance. The foundation of this method is based on the so-called *labeled Boolean networks* and *configurations*.

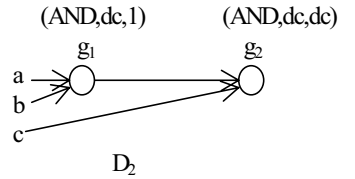
A local configuration is a sub-network, which reflects the structural relationship of the represented circuit. Clearly, the minimal configurations containing the alternative wire are considered as the minimal sub-structures of Boolean network containing alternative wires. Let's use an example to illustrate the Boolean network, sub-network, configuration and their relations. In Fig. 2(a), G is a Boolean network, and S is a sub-network. In Fig. 2(b), D_1 shows one graph configuration of S. In D_1, a node is denoted as a triple (op, r, s). Here op is the operator representing the Boolean function of g, while r and s are nonnegative integers. All the edges inside S are kept unchanged, but the edges outside the S are omitted. Usually, r equals to $d^-(g)$ and s equals to $d^+(g)$. In order to apply the configuration to more sub-networks, we define that op, r and s can have the don't care (*dc*) value. That is, dc can represent any logic operator, any positive number of input and any positive number of output. Fig. 2(c) gives another configuration of sub-network S. Both D_1 and D_2 can match with the S sub-circuit although D_2 is more general and covers more circuit patterns.



(a) Boolean network G



(b) A configuration of S



(c)Another configuration of S

Fig. 2 Configuration of sub-network

In a Boolean network, the level of a node g, denoted by l(g), is defined as the *maximum path length* from primary inputs to g. For example, in Fig. 2, $l(g_1)=1$, $l(g_2)=2$, $l(g_3)=3$. The distance, k, of two wires, $<g_1,g_2>$ and $<g_3,g_4>$, is defined as $k = |l(g_4) - l(g_2)|$.

For simplicity, in this paper, we will not differentiate among Boolean network, circuit and graph (or hypergraph) [17]. We only consider circuits (Boolean networks) with AND, NAND, OR, INV and NOR simple gates. Complex gates can be handled by decomposing them into these simple gates.

3 ATPG-based technique

Mandatory assignments (MA) are the unique values certain nodes have to hold for a test pattern to exist. For a given stuck-at fault f, the set of mandatory assignments, denoted as SMA(f), can be computed with the technique in [18]. A suck-at fault f is untestable, and therefore redundant, if its corresponding SMA(f) can not be consistently justified.

A wire w_t is redundant if its corresponding stuck-at fault is untestable. The ATPG-based alternative wire technique is an alternative wire locating technique, which identifies alternative wires of a target wire through ATPG (Automation Test Pattern Generation) methods. Regarding a target wire w_t, if its stuck-at fault becomes untestable when another redundant wire w_a is added, then wire w_a is an alternative wire of w_t. That is, adding wire w_a can then makes wire w_t removable without changing the functionality of the circuit.

An ATPG-based rewiring technique, in general, first computes the MA for the target wire, w_t, stuck-at fault test. Secondly, it collects a set of candidate connections that can make the target wire redundant by making the targeted MA inconsistent with the addition of the candidate wire. Then, it checks if this candidate connection is redundant. If it is redundant, then it is an alternative wire for the target wire; otherwise, it is not. In this stuck-at test procedure, logic implication process has to be conducted. If the techniques only conduct simple logic implications, we only obtain the MA partially, then some potential alternative wires may be omitted. If we use some complex logic implication techniques, such as *recursive learning*[18], then the computation can be even more intensive. Generally, all these techniques need to apply Boolean operations and are prone to be time consuming.

In order to reduce the CPU run time of the ATPG-based techniques, some improvements were proposed in [13]. ATPG-based rewiring technique has been widely used in solving some logic synthesis problems, it is detailed in [10-15].

4 Graph-based Alternative Wire Technique (GBAW)

The Graph-Based Alternative Wire (GBAW) is a newly proposed rewiring technique. It searches alternative wires by checking graph matching between local sub-networks and the pre-specified minimal sub-graph configurations containing alternative wires within a given distance. A configuration is a minimal circuit pattern containing alternative wires. When a sub-network matches a pattern, we can quickly determine its target wire and corresponding alternative wires. Obviously, if w_r is an alternative wire of w_t, then w_t is also an alternative wire of w_r. In a pattern, both w_r and w_t are present. But in the sub-network, only one of them is existent (we assume the network under processing contains no redundant wires). In a pattern, if there exists a wire $w_t <x_1,y_1>$ and its alternative wire $w_r <x_2,y_2>$, and k is the distance of these two wires, then we call such a pattern as a k-local pattern. Fig. 3 gives three 2-local patterns used in GBAW. In Fig. 3, we show the target wire and the alternative wire as a thick line and a dashed line respectively (Their roles can be exchanged).

We note that the notion of applying local pattern matchings was also used in the rule-based expert system [19]. However, the rule-based expert system iteratively replaces and rearranges small portions of a circuit with the purpose of minimizing circuit areas. The purpose of GBAW, however, is to find the alternative wires of the target wire within a limited distantance, and is able to locate remote alternative wires in a waveform propogation.

In the implementation of GBAW, we considered 9 basic patterns in total. Not only can GBAW handle the case of adding one wire and removing another wire, it can also process the cases of adding one AND gate, OR gate, NAND gate or NOR gate in order to remove the target wire as well. It has also considered the case of simultaneously adding two gates to remove the target wire and the case of adding one wire to simultaneously removing two wires. Due to space limitation, we do not discuss the details here.

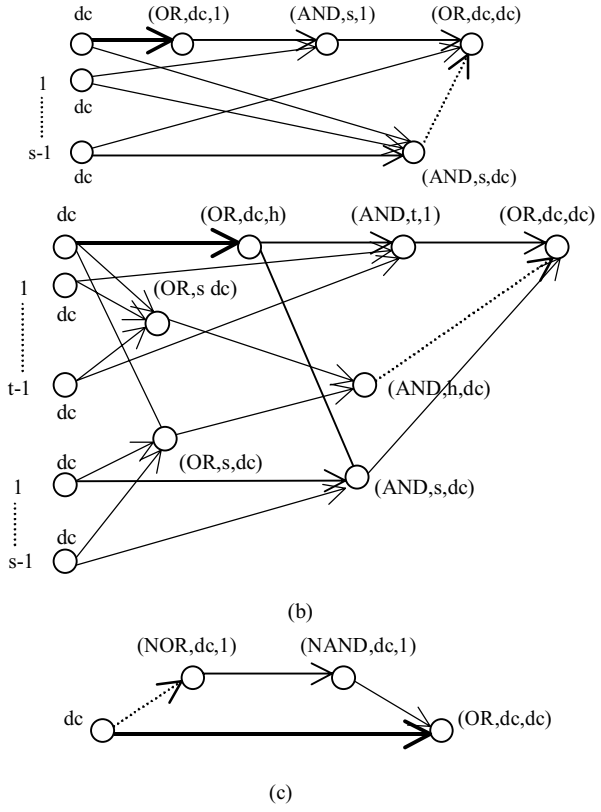


Fig. 3 Examples of 2-local patterns

5 Repartitioning through Rewiring

A *wire* here is defined to be a 2-point connection between a pair of source and sink nodes. When a larger circuit is partitioned into two sub-circuits, we define the wires across the partitioning cut line as cut wires. Fig. 4 gives an example of cut wires. In Fig. 4, a larger circuit C is partitioned into two sub-circuits C_1 and C_2, wires <a,c>, <a,e>, <b,d> and <b,e> are all cut wires. We can see, there are only two nets (hyperedges [17]) between the two sub-circuits. One is the a-net (it includes wires <a,e> and <a,c>), the other is b-net (it includes wires <b,d> and <b,e>). We also call such two nets as cut nets. Obviously, it needs two pins in each sub-circuit in order to connect these two sub-circuits. The objective of partitioning is to minimize the pins required to connect the sub-circuits, that is, to minimize the number of cut nets. Because some of the wires in a circuit may have alternative wires, if we replace some of the cut wires by its alternative wires that are not cut wires, then the number of cut net may be reduced. Additionally, this rewiring will also lead to a new circuit modeling graph, which may drag the partitioning out of a local minimal. This is the main idea of applying alternative wire technique for repartitioning in this paper.

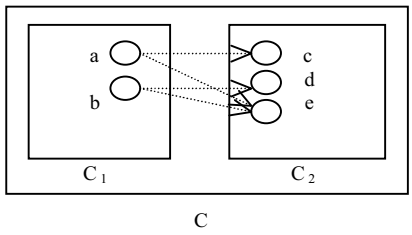
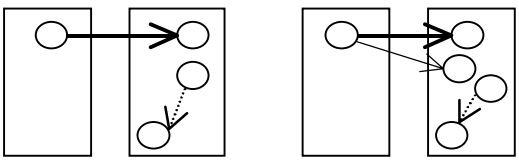
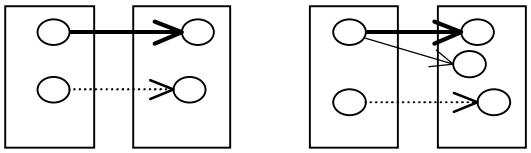


Fig. 4 An example of cut wires

We notice that if we put no restriction on the alternative wires, the number of pin required to connect sub-circuits may increase, reduce or keep unchanged when a cut wire is replaced by its alternative wire. In Fig. 5, the gain value reflects the change of cut-cost, the number of (cut) pins. If the number of pins is reduced, a positive integer is assigned to the gain. Similarly, a zero (negative) integer is assigned if the number of pins is unchanged (increased). The absolute value of the integer is equal to the number of variation of pins. In Fig. 5(a), the number of pins is reduced by 2. In Fig. 5(b) and Fig. 5(c), the number of pins keeps unchanged. In Fig. 5(d), the number of pins is increased by 2.



(a) gain=2 (b) gain=0



(c) gain=0 (d) gain=-2

Fig. 5 Variation of the number of pin after circuit restructuring

We select the well-known FM partitioning algorithm [17] to produce a good partition first, then we apply RG to perform structure perturbation for further improvements. The perturbation operations include: substituting a wire with its alternative wire, adding a gate and removing some wires, adding one wire to remove two wires, adding two gates to remove one wires, … etc. Fig. 6 gives the basic repartitioning sub-procedure Sub-R of our algorithm RG.

```
Procedure Sub-R (best_partition, PT, k, t) {
1.   tag = true;
2.   exit = false;
3.   search_limit = 0;
4.   n_perturbations = 0;
5.   curr_partition = best_partition;
6.   last_partition =  best_partition;
7.   while ((n_perturbations < k) && (exit = = false)){
8.      search_limit = 0;
9.      while (search_limit < t) {
10.        search_limit + +;
11.        randomly select a cut wire Wt;
12.        use PT to find all alternative wires SWa for Wt;
13.        if (SWa = = ϕ) {
14.           search_limit + +;
15.           continue;
17.         } else
18.             break;    }
19.     if (SWa ! = ϕ) {
20.        pick alternative wire W1 with the largest
                                                  gain;
21.        replace Wt with W1 in curr_partition;
22.        curr_partition = FM(curr_partition);
23.         n_perturbations = n_perturbations + 1;
24.       if (cost(curr_partition) < cost(last_partition))
25.         last_partition = curr_partition; }
27.     else
28.        exit = true;
29.     }
30.  }
```

Fig. 6 Procedure Sub-R of logic perturbation for partitioning

During the perturbation process of Sub-R, we only select cut wires as the target wires for conducting perturbation as the same as in [9]. We first randomly select a cut wire as the target wire. Secondly, we use PT (Perturbation Technology) in finding the alternative wire set SWa of the target wire. Here, PT can be RAMBO or GBAW. Finally, we select one of the alternative wires from the SWa which has the highest gain for circuit perturbation. When the SWa of the selected

cut wire is empty, we may randomly select another cut wire to try again. The number of times of this try should not exceed t times. Similarly, k sets the limit of perturbations. These limit settings can save a lot of time when the total number of the alternative wires of all cut wires is 0 or very small. The main difference of procedure Sub-R and the algorithm in [9] lies on the condition of perturbation. In [9], perturbation will be performed only when the alternative wire of the selected cut wire has a non-negative gain. However, from our experimental results, we find that sometimes a better result could be found if we allow for some logic perturbations with negative gain in cut size. In this paper, a perturbation is always be done regardless if the gain of the alternative wire is non-negative or not. This can help the perturbation process jump out of local optimality.

Different order of applying PTs (GBAW and RAMBO) can bring different results. Fig. 7 gives how RG manupulates this situation. When the parameter s of RG is 1, only the RAMBO technique is used to do repartitioning. When the parameter s of RG is 0, both RAMBO and GBAW are used to do repartitioning. Depending on the order of applying RAMBO and GBAW, two repartitioning results are produced (last_partition1 and last_partition2). Finally, we select the best of the two results (last_partition1 and last_partition2) as the final partition when s is set to 0.

```
Algorithm Repartitioning through Logic-perturbation
                                                (s, n, k, t) {
1.      Perform FM n times; save the best parti-
          tion as best_partition;
2.      if (s = = 0) {
3.          Sub_R(best_partition, RAMBO,k,t);
4.          best_partition1 = last_partition;
5.          Sub-R((best_partition1, GBAW,k,t);
6.          last_partition1 = last_partition;
7.          Sub_R(best_partition, GBAW,k,t);
8.          best_partition1 = last_partition;
9.          Sub-R((best_partition1, RAMBO,k,t);
10.         last_partition2 = last_partition;
11.         last_partition = min (last_partition1  ,
            last_partition2);
12.     }
13.     else if (s = = 1) {
14.         Sub_R(best_partition, RAMBO,k,t);
15.     }
16. }
```

Fig. 7 Algorithm of RG

6 Experimental Results

We have implemented algorithm RG in C language and conducted experiments on a Sun Ultra 5/270 workstation for MCNC benchmarks, ranging from medium to large in circuit size. These circuits have been simplified by SIS script "script.algebraic" and mapped by SIS command "map" with "mcnc1.genlib", a sub-set of "mcnc.genlib", which includes simple gates and limits the gate fanins to 2. The RAMBO version shown in [12, 14] is used as the ATPG-based rewiring technique in RG. (In the RAMBO version we run, we have also included the cases of adding gates to find alternative wires of the target wire.)

Table 1 lists the number of nodes and the number of wires of the MCNC benchmark circuits we experimented. Table 2 lists the statistics of alternative wires on these circuits. In Table 1 and Table 2, Column "Circuit" shows the name of the circuit under experiments. Column "Node" lists the number of nodes (including input node, internal node and the output node) of the circuit. Column "Wire" lists the number of wires. Column "alt. wire" lists the number of alternative wires found for all of the wires in the circuit and Column "%" lists the percentage of these wires with respect to the total number of wires. Column "CPU" lists the CPU time in terms seconds. From the experimental data listed in Table 2, we can see that although GBAW has found more alternative wires than RAMBO on average, but it uses very little CPU time. Probably due to the avoidance of running logic implications, GBAW can run faster than currently known ATPG-based techniques in finding local alternative wires.

In order to show the power of RG in logic perturbation, we set the tolerance of area imbalance on the FM method to be ±20% of the average area in each partitioned block. Then we ran our FM program for 250 times on each circuit as in [9], thus we may be able to get a quite good, if not optimal, set of partitioning solutions. We select the best solution and use RG for logic perturbation to further improve the quality of the partitioning with n =250, k = 60, and t = 50. Table 3 and Table 4 list the experimental results of FM only and with RG(s = = 1) respectively. RG(s = = 1) means only RAMBO is used. Table 5 lists the experimental results of RG(s = = 0). In RG(s = = 0), both the RAMBO and GBAW are coupled for circuit perturbations. Finally, we select the best results. Column "area" lists the area of the sub-circuit. Column "cut cost" lists *the total number of cut pins* obtained by FM algorithm. Column "cut wire" lists the number of cut wires of the partitioning. Column "cut cost" in table 4 lists the partitioning cost obtained by RG(s = 1) and Column "%" lists the percentage of the cost reduction. Column "cut cost" in table 5 lists the partitioning cost (in terms of the total number of cut pins) obtained by RG(s = 0) and Column "%" lists the percentage of the cost reduction. From Table 4, we can see that using RAMBO only for perturbation, we can obtain on average 10.9% of improvement over the best partition of 250 times of FM. From Table 5, we can see that through logic perturbation of RAMBO and GBAW (RG(s = = 0)), we can obtain on average 16.2% of improvement over the best partition of 250 times of FM, i.e., which is much better than applying RAMBO alone without much additional CPU overhead.

Circuit	Node	Wire
C432	245	399
C3540	1285	2289
alu4	793	1478
des	4084	6900
alu2	428	783
C5315	2085	3405
C1355	632	1087
C7552	2530	4213
term1	282	449
rot	931	1358
TOTAL	13295	22361

Table1: Node number and wire number of the circuit

Circuit	GBAW		RAMBO	
	alt. wire(%)	CPU (sec)	alt. wire(%)	CPU (sec)
C432	254(63%)	0.28	238(59%)	19.01
C3540	1222(53%)	1.42	1039(45%)	579.55
alu4	576(38%)	0.91	597(40%)	448.76
des	3384(49%)	6.51	2643(38%)	2680.42
alu2	312(39%)	0.5	313(39%)	229.19
C5315	787(23%)	2.01	728(21%)	301.63
C1355	250(22%)	0.61	178(16%)	37.42
C7552	685(16%)	3.12	902(21%)	631.86
term1	202(44%)	0.25	191(42%)	22
rot	440(32%)	0.81	402(29%)	66.16
TOTAL	8112	16.42	7231	5016
AVG	(37.9%)	1.642	(35%)	501.6

Table 2: Alternative statistics

Although it is difficult to directly compare the experiment results between ours and [9], due to the difference of circuit inputs. The coupled perturbation scheme, under our experimental set up, can consistently perform better, which indicates that such a coupling can exploit a larger domain of useful guided circuit perturbation effectively and efficiently.

	FM			
model	area	cut cost	cut wire	cpu(sec)
C432	126:112	28	46	163.2
alu2	195:227	88	126	313
C7552	1450:972	228	401	3292.8
alu4	461:324	160	298	612.2
C3540	743:520	180	261	1300.7
C5315	793:1169	234	413	2214.1
C1355	242:358	46	62	487.3
rot	342:482	66	80	623.5
term1	129:143	28	24	166.5
des	2222:1617	332	964	4475.2
TOTAL		1390	2675	13648.5

Table 3: 2-way partitioning by FM

	RG(s = = 1)			
model	area	cut cost(%)	cut wire	cpu(sec)
C432	112:143	24(14.2%)	24	42.1
alu2	205:253	88(0%)	110	780.4
C7552	1406:1024	164(28%)	328	357
alu4	469:350	146(8.7%)	232	1534.8
C3540	755:523	174(3.3%)	248	713.5
C5315	794:1172	210(10.2%)	346	36.7
C1355	246:362	46(0%)	58	24.9
rot	485:354	58(12.1%)	52	61.4
term1	158:131	20(28.5%)	12	72
des	1574:2306	316(4.8%)	939	1153.8
TOTAL		1246	2349	4776.6
AVG		(10.9%)		

Table 4: 2-way partitioning by RG(s==1, RAMBO only)

	RG(s = = 0)			
model	area	cut cost	cut wire	cpu(sec)
C432	134:115	20(28.5%)	24	90.4
alu2	201:252	84(4.5%)	133	968.9
C7552	1408:1043	164(28%)	328	858.6
alu4	457:361	140(12.5%)	232	3199.9
C3540	749:537	148(17.7%)	236	1094.1
C5315	797:1165	200(14.5%)	330	171.1
C1355	344:266	42(8.6%)	54	51.1
rot	482:354	56(15.1%)	50	159.4
term1	154:131	20(28.5%)	12	146.9
des	1577:2312	316(4.8%)	935	1823.1
TOTAL		1190	2334	8563.5
AVG		(16.2%)		

Table 5: 2-way partitioning by RG(s==0, RAMBO and GBAW)

7 Conclusion

The recently developed alternative wiring technique has been shown to be very useful in many EDA problems. The ATPG-based rewiring technique is more flexible; however the graph-based rewiring technique is faster and can couple quite well with the ATPG-based scheme to exploit a much larger room for logic perturbations.

In this paper, we have proposed an RG algorithm that couples both rewiring techniques to obtain better partition results without paying high CPU cost. The scheme shown has been able to further improve the nearly best results produced by the conventional FM algorithm, which justifies the practical value of the proposed partition scheme. Since GBAW is very fast, it is possible to include more patterns to locate alternative wires without increasing the CPU cost significantly. This could be our next step for further improvement.

For our future research, the circuit perturbation scheme proposed in this paper could also be applied to solve other problems lied in IC design automation, especially for very large circuits.

References

[1] J. Hwang and A. El Gamal, ”Optimal Replication for Min-Cut Partitioning,” ICCAD-92, pp. 432-435.

[2] Y. C. Wei and C. K. Cheng, ”Ratio Cut Partitioning for Hierarchical Designs,” IEEE Tran. On CAD, July 1991, pp. 911-921.

[3] C. W. Yeh, C. K. Cheng, and T. T. Y. Lin, ”A probabilistic Multicommodity-Flow Solution to Circuit Clustering Problems,” ICCAD-92, pp. 428-431.

[4] L. Hagen and A. B. Kahng, ”Fast Spectral Methods for Ratio Cut Partitioning and Clustering,” ICCAD-91, pp. 10-13.

[5] C. Kring and A. R. Newton, ”A Cell-Replicating Approach to Mincut-Based Circuit Partitioning,” ICCAD-91, pp. 2-5.

[6] M. Beardslee, B. Lin, and A. Sangiovanni-Vincentelli, ”Communication Based Logic Partitioning,” EDAC-92, pp. 32-37.

[7] D. I. Cheng, S. C. Chang, and M. Marek-Sadowska, ”Partitioning Combinational Circuits in Graph and Logic Domains,” Proc. SASIMI-93, pp. 404-412.

[8] R. Kuznar, F. Brglez, and B. Zajc, ”Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect,” DAC-94, pp. 238-243.

[9] D. I. Cheng,C. C. Lin, and M. Marek-Sadowska, ”Circuit Partitioning with Logic Perturbation,” in Proc. ICCAD’95, 1995, pp. 650-655.

[10] S. C. Chang, K. T. Cheng, N. S. Woo, and M. Marek-Sadowska, ”Layout Driven Logic Synthesis for FPGA,” in Proc. DAC, June 1994, PP. 308-313

[11] S. C. Chang and M. Marek-Sadowdka,” Perturb and Simplify: Multi-level Boolean Network Optimizer,” in Proc. ICCAD, Nov. 1994, PP. 2-4.

[12] K. T. Cheng and L. A. Entrena, ”Multi-level Logic Optimization by Redundancy Addition and Removal,” in Proc. Europ. Conf. Design Automation, Feb. 1993, pp. 373-377.

[13] S. C. Chang, L. P. V. Ginneken, and M. Marek-Sadowska, ”Fast Boolean Optimization by Rewiring,” In proc. ICCAD’96, 1996, pp. 262-269.

[14] S. C. Chang, M. Marek-Sadowska, and K-T Cheng, ”Perturb and Simplify: Multilevel Boolean Network Optimizer,” IEEE Trans CAD of ICAS, Vol. 15, No. 12, Dec 1996, PP. 1494-1504.

[15] S. C. Chang, K. T. Cheng, N-S Woo, and M. Marek-Sasowska, ”Postlayout Logic Restructuring Using Alternative Wires,” IEEE Trans. CAD of ICAS, Vol. 16, No. 6, June 1997, pp. 587-596.

[16] Y. L. Wu, Hongbing Fan, ”On Local Configuration Analysis of Alternative Wires in Boolean Networks” ITC-CSCC’99 pp. 868-871.

[17] C. M. Fiduccia and R. M. Mattheyses, ”A Linear Time Heuristic for Improving Network Partitions,” DAC-82, pp. 175-181.

[18] W. Kunz and D. K. Pradham, ”Recursive Learning: An Attractive Alternative to the Decision Tree for Test Generation for Digital Circuit,” Int. Test Conf., 1992, pp. 816-825.

[19] David Gregory, Karren Bartlett, etc, "Socrates: A System for Automatically Synthesizing and Optimizing Combinational Logic," in 23rd Design Automation Conference, 1986, PP.79-85.