Simultaneous Circuit Transformation and Routing

Hiroaki Yoshida Motohiro Sera Masao Kubo Masahiro Fujita

Department of Electronic Engineering
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: hiroaki@silicon.u-tokyo.ac.jp

Abstract

In this paper, we propose a new methodology to integrate circuit transformation into routing. More specifically, this paper shows an approach for performing routing and wire reconnection simultaneously. To accomplish this, we introduce a new logic representation that implements all possible wire reconnections implicitly by enhancing global flow optimization techniques. Since our method takes into account circuit transformation during routing phase where the accurate physical information is available, we can obtain better results than the conventional routing algorithms. In addition, we can succeed in routing even if other routers like rip-up and reroute methods fail. The algorithm has been implemented and the experimental results are presented. We believe this is the first approach which combines them completely.

1. Introduction

As feature sizes decrease and chip sizes increase, the area and performance of chips become dominated by the interconnect. Since it is difficult to obtain the physical implementation until routing phase, most logic synthesis tools cannot estimate the effects of the interconnect accurately. In addition, when some nets cannot be routed properly or design doesn't satisfy constraints in routing phase, we have to perform logic synthesis again.

Logic optimization algorithms based on rewiring techniques[1] have been developed in the last decade. The basic idea of these algorithms is to add and remove redundant wires in a circuit. Due to the incremental nature, the techniques can easily be applied to a circuit even after its placement is fixed. For performance optimization, Jiang *et al.* proposed a post-layout logic restructuring technique based on the rewiring technique[2]. It picks one of the rewiring , and then performs routing incrementally. In this approach, wire reconnection and routing are performed independently.

In this paper, we present an approach for performing routing and wire reconnection simultaneously. To accomplish this, we introduce a new logic representation that implements all possible wire reconnections implicitly. Since our method takes into account circuit transformation during routing phase where the accurate physical information is available, we can obtain better results than the conventional routing algorithms.

The rest of this paper is organized as follows. In the next section, we briefly review global flow optimization and exact routing methods. In Section 3, we introduce a multi-level logic representation of an implication flow graph and show how to derive all solutions from the representation. Section 4 describes our approaches to unify routing and wire reconnection. Experimental results are presented in Section 5.

2. Overview of Existing Algorithms

2.1. Global Flow Optimization

In [3], global flow optimization technique has been proposed. It gathers a circuit information using techniques of data flow analysis, and then reconnect the immediate fanouts of a node to the inputs of other nodes (so called fanout global flow optimization). Recently, Chang *et al.* have shown that the method cannot fully characterize a circuit and hence the optimality may be lost[4]. They have also introduced a modified technique using new graph, which is called an implication flow graph.

We briefly review the global flow optimization method using the implication flow graph. It is assumed that a circuit consists of only NOR gates. It first assigns a target node s to a value 1 and then propagate it towards the primary outputs. Figure 1 illustrates the value assignments of an example circuit, where the target node is the node s. Next, an implication flow graph is constructed as follows.

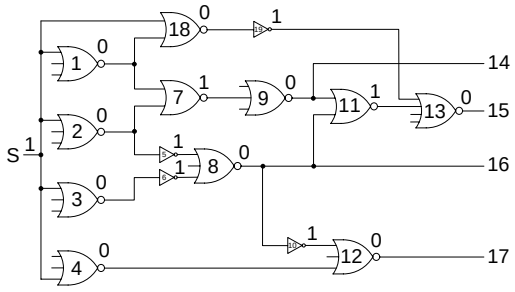

Figure 1. Example circuit.

1. Add a source node *Src* representing the node *s* and a sink node *Snk*.

2. Add a corresponding node for a node with implication. If a node has a controlling value, the corresponing node in the implication flow graph has an IMP_OR type with weight 1. If a node has a non-controlling value, the corresponding node has an IMP_AND type with weight infinite.

3. Edges are added as follows. First, the IMP_AND node is connected to all its corresponding input nodes. An IMP_OR node is connected to all its corresponding nodes with controlling value to the node. Add the edges from all the frontier nodes to the sink node.

After building the implication flow graph, a degenerated implication flow graph is constructed by removing all but one fanin edges for each IMP_OR node. Any cutset in the degenerated implication flow graph can form a solution for the fanout reconnection. Figure 2 shows one of the degenerated implication flow graphs derived from the circuit shown in Figure 1. Since the node 8 and 9 form a cutset of the graph, we can reconnect the immediate fanouts of the node *s* to the inputs of the node 8 and 9 in Figure 1. The resulting circuit is shown in Figure 3. Obviously, the quality of the solution heavily depends on the way how to remove fanin edges from the implication flow graph and how to obtain a cutset in the graph.

2.2. Exact Routing Using Symbolic Representation

In [5], Schmiedle *et al.* have presented an exact approach for solving channel routing problems. They have also shown a search space reduction technique[6]. They use Multi-valued Decision Diagrams(MDDs) for representation of the routing space. All possible solutions are represented in a single MDD. Figure 4 shows an example of MDD. The function f represented by this MDD satisfies $f = 1$ only when $(a, b, c, d) = (0, 0, 2, 1), (1, 0, 0, 2), (1, 0, 2, 0)$.

We briefly review the exact routing method. The method assumes that the grid-based model is used, *i.e.*, the routing

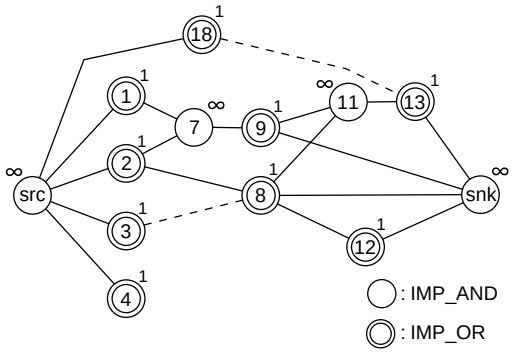

Figure 2. Degenerated implication flow graph.

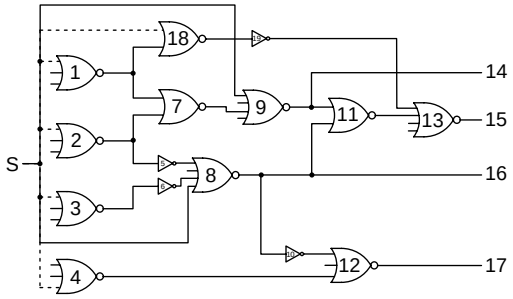

Figure 3. Example circuit after wire reconnection.

region are divided to rectilinear grids. In routing region, x-direction is referred to *columns*, y-direction to *tracks* and z-direction to *layers*. Every grid point is represented by an MDD variable m_{xyz} ($x = 1, ..., length, y = 1, ..., width, z = 1, ..., height$). Given nets $N = \{N_1, ..., N_n\}$ to be routed, the set of legal values for these variables is $\{0, ..., n\}$. $m_{xyz} = k$ means that the routing grid at (x, y, z) is occupied by net N_k and $m_{xyz} = 0$ means that the routing grid at (x, y, z) is not occupied by any net.

For constructing the MDD representation corresponding to the set of all solutions, *conectivity predicates* are computed first by a fixed point iteration. A connectivity predicate $C_{i,t}(p)$ indicates that p is connected with t via net N_i. An iteration starts with $C^0_{i,t}(t) = (t = i)$ for a terminal t and $C^0_{i,t}(p) = false$ for all $p \neq t$. In jth iteration, $C^j_{i,t}(p)$ for each grid point p is computed by the following equation

$$C^j_{i,t}(p) = C^{j-1}_{i,t}(p) \vee (\bigvee_{p'} (C^{j-1}_{i,t}(p') \wedge (m_{xyz} = i))) \qquad (1)$$

where p' and p are adjacent. The iteration is continued until a fixed point reaches. Then connectivity predicate $C_{i,t}(p) = C^{j_{max}}_{i,t}$ is obtained. In other words, we can view this process as exploring the grid in every direction like Lee's maze router [7] and recording all possible paths.

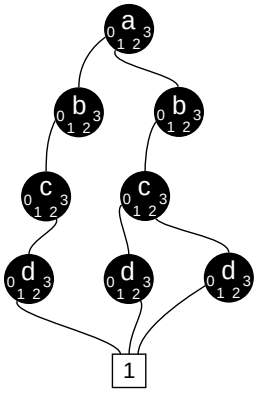

Figure 4. Multi-valued Decision Diagram (MDD).

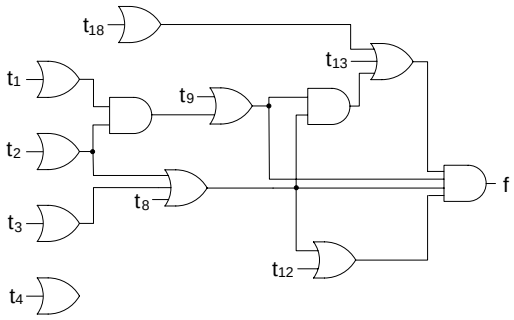

Figure 5. Logic representation of the implication flow graph.

An MDD n_i representing all possible paths of net N_i can be derived from the equation:

$$n_i = \bigwedge_{j=2}^{|N_i|} C_{i,t_{i1}}(t_{ij}) \tag{2}$$

where t_{ij} is the jth terminal of net N_i. Finally, the MDD representing routing solutions for all nets is obtained by combining all n_i.

$$s = \bigwedge_{N_i \in N} n_i \tag{3}$$

A given routing problem is solvable if and only if $s \neq 0$.

3. New Logic Representation of Implication Flow Graph

In this section, we introduce a new logic representation of the implication flow graph. Since this representation contains all possible wire reconnections implicitly, we don't have to care about how a degenerate implication flow graph is obtained. In addition, it enables us to solve a global flow optimization problem efficiently.

We can derive a logic representaion from a given implication flow graph as follows.

1. Transform IMP_AND and IMP_OR nodes into AND and OR gates respectively.
2. Replace *Snk* node with AND gate.
3. Remove *Src* node and its fanout edges.
4. Create new Boolean variables which correspond to each OR gate. Each variable is inserted as the corresponding OR gate's fanin.

Let $t_1, ..., t_n$ be the new Boolean variables. Then the resulting Boolean function is represented by $f(t_1, ..., t_n)$. Each input pattern which satisfies the function f corresponds to a wire reconnection. Figure 5 shows the logic representation of the implication flow graph in Figure 2. For instance, $(t_8 = 1, t_9 = 1)$ satisfies the function f, then we can reconnect the immediate fanouts of the node s to the inputs of the node 8 and 9 in Figure 1.

As can easily be seen, each prime implicant of the Boolean function corresponds to a cutset of a degenerated implication flow graph. Since the function is a unate function, all prime implicants are essential and will appear in the minimized function[8]. Therefore we can use many existing two-level logic minimization algorithms in order to obtain the solutions of a global flow optimization problem. In the case of Figure 5, the function f is minimized to:

$$f = t_1 \cdot t_2 + t_2 \cdot t_9 + t_3 \cdot t_9 + t_8 \cdot t_9 \tag{4}$$

and so possible rewires are $\{1, 2\}$, $\{2, 9\}$, $\{3, 9\}$, $\{8, 9\}$, *i.e.*, the immediate fanouts of the node S in Figure 1 can be reconnected to node 1 and node 2, or node 2 and node 9, and so on.

4. Simultaneous Wire Reconnection and Routing

In this section, we present an exact approach for solving the reconnection and routing problems simultaneously. Since both problems have already been transformed into Boolean functions, unification of these two algorithms can be easily accomplished.

Our layout model is illustrated in Figure 6. There is a routing region between two parallel rows of cells with terminals at the top or bottom of the routing region. Some cells have extra terminals and the nets can be reconnected to them. It is assumed that the cells are replaced with the appropriate cells if the reconnections are performed.

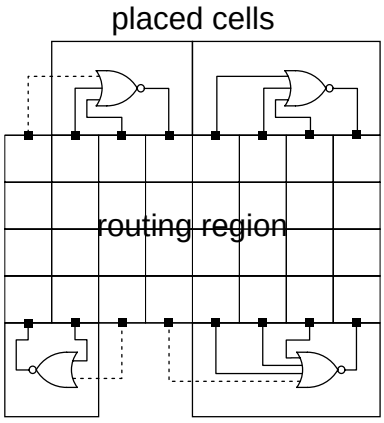

Figure 6. Layout model.

At first, we create the logic representation of implication flow graph for the net N_i and then obtain the Boolean formula $f(t_{i2}, ..., t_{in})$ where $t_{i2}, ..., t_{in}$ are Boolean variables corresponding to the candidate terminals. To combine this Boolean formula with an exact routing, we just replace the Equation (2) with:

$$n_i = f(C_{i,t_{i1}}(t_{i2}), ..., C_{i,t_{i1}}(t_{in})) \quad (5)$$

Note that we can use the multi-level logic representation of the function f to calculate the Equation (5), rather than the two-level logic representaion. The resulting MDD represents all possible routing solutions with consideration of wire reconnection. That is, in this framework, logical rewiring and detail routing are done simultaneously. We believe this is the first approach which combines them completely.

As shown in the original paper[5], the exact router can handle only small problems because the size of the MDDs become too large. If one's objective is just to obtain one of the solutions, we can solve larger problems. We evaluate the Equation (3) for each connectivity predicate calculation step, and stop the calculation when any solutions are found. Using this technique, we can hope that the solutions are found before the MDDs blow up.

5. Experimental Results

The technique presented in Section 3 was implemented and we performed experiments on MCNC91 benchmark circuits. The circuits are first decomposed to only NOR gates and then the technique is applied to each wire. Each logic representation of implication flow graph is collapsed to two-level logic and minimized by ESPRESSO[8]. The number of the wire reconnections corresponds to the number of the product terms in the minimized expression. The results are shown in Table 1. The third column shows the number of the nets with one or more wire reconnections. In column 4, the cumulated sum of the number of the wire reconnections for each net is shown. Note that the original connections are excluded from the wire reconnections. The results show that there are many wire reconnections in the circuits. For some circuits which are not shown in Table 1, such as C2670 and C3540, we couldn't obtain the results because the resulting two-level expressions became too large. This may be avoided by using the implicit prime implicants enumeration methods like [9].

Table 1. Number of wire reconnections.

circuit	*#nets*	*#rec. nets*	*#rec.*	*CPU [sec.]*
C432	283	63	108	1.0
C499	604	18	78	2.9
C880	532	112	307	5.1
C1355	644	122	206	3.1
C1908	671	70	174	3.8
C5315	2144	194	228	10.2
C6288	2432	435	435	13.0
C7552	3219	272	384	25.1
alu2	284	31	196	8.0
alu4	547	30	1038	8.9
b9	209	57	142	0.7
c8	230	45	302	1.0
cc	120	33	80	0.3
cm150a	101	42	42	0.3
cmb	68	32	64	0.2
comp	218	86	102	1.0
cu	80	28	425	0.4
lal	202	47	6986	12.5
mux	81	36	36	0.3
pair	2154	604	1228	10.5
pm1	93	21	137	0.2
rot	854	43	207	2.1
sct	171	19	70	0.5
term1	633	325	1217	7.8
x1	448	23	75	4.3

Next, the algorithm presented in Section 4 and [5] was implemented and we performed experiments on some example circuits. We consider an example circuit that consists of three nets and a routing grids with a channel that has 6 columns, 4 tracks and 2 layers as shown in Figure 7. The Boolean functions that are derived from the method described in Section 3 are as follows:

$$f_x = (b + d)eg \quad (6)$$
$$f_y = cfh \quad (7)$$
$$f_z = ai \quad (8)$$

In Equation (6)-(8), the Boolean functions f_x, f_y, f_z correspond to the terminals x, y, z in Figure 7 respectively and

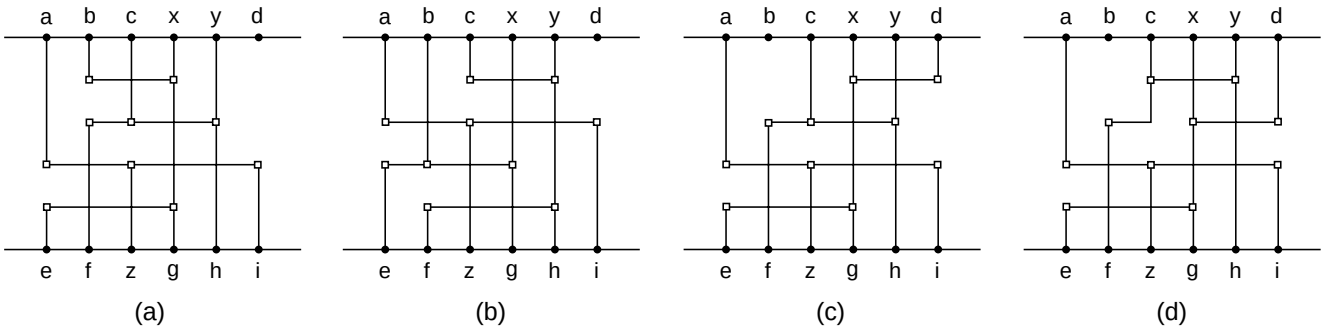

Figure 7. Routing solutions.

Table 2. Experimental results with the heuristic technique.

name	*#grids*	*#nets*	*#iter.*	*memory (MByte)*	*CPU (sec)*
example1	$6 \times 4 \times 2$	3	9	4	1.8
example2	$7 \times 5 \times 2$	3	10	30	19.9
example3	$8 \times 6 \times 2$	3	8	276	281.0

each Boolean variable corresponds to the terminal in Figure 7. That is, the terminal x can be connected to the terminals b, e, g or the terminals d, e, g. The terminal y must be connected to terminals c, f, h and the terminal z to the terminals a, i. We were able to find all possible solutions within 800 seconds. Figure 7 shows some of the solutions. In Figure 7 (a)(b), the terminal x is connected to the terminals b, e, g, and the terminal x is connected to the terminals d, e, g in Figure 7(c)(d).

For larger examples, the heuristic technique described in Section 4 was applied. Equation (3) in Section 2.2 are evaluated for each connectivity predicate calculation step and the calculation is stopped when any solutions are found. Table 2 shows some results for some example circuits up to 8 columns, 6 tracks, 2 layers and 3 nets. The fourth column shows the number of the evaluations of Equation (3). In column 5 and 6, memory usage and CPU time are shown respectively.

6. Conclusions and Future Work

In this paper, we first introduced a logic representation of an implication flow graph. This representaion enables us to solve the global flow optimization problem efficiently. Then, we showed an exact approach for performing routing and wire reconnection simultaneously. Although the approach can handle only small portions of circuits, it allows us to explore larger solution space than the previous routing methods. In the future, we plan to integrate other routing algorithms to handle practical problems.

As mentioned earlier, the global flow optimization technique assumes that circuits consist of only NOR gates. Therefore our method cannot handle general circuits containing complex gates such as XOR, AND-OR-INVERTER, *etc*. An extension to general circuits is currently being investigated.

References

[1] K.-T. Cheng and L. A. Entrena. Multi-level logic optimization by redundancy addition and removal. In *in Proc. ACM/IEEE European Conf. Design Automation*, pages 373–377, Feb. 1993.

[2] Y. M. Jiang, A. Krstic, K. T. Cheng, and M. Marek-Sadowska. Post-Layout Logic Restructuring for Performance Optimization. In *Proc. ACM/IEEE Design Automation Conf.*, pages 662–665, June 1997.

[3] C. L. Berman and L. H. Trevillyan. Global flow optimization in automatic logic design. *IEEE Trans. Computer-Aided Design*, 9(5):557–564, May 1991.

[4] S. C. Chang, Z. Z. Wu, and H. Z. Yu. Wire Re-Connections Based on Implication Flow Graph. In *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, pages 533–536, Nov. 2000.

[5] F. Schmiedle, R. Drechsler, and B. Becker. Exact Channel Routing Using Symbolic Representation. In *Proc. IEEE Int. Symp. Circuit and Systems*, pages 394–397, May 1999.

[6] F. Schmiedle, D. Unruh, and B. Becker. Exact Switchbox Routing with Search Space Reduction. In *Proc. ACM Int. Symp. Physical Design*, pages 26–32, Apr. 2000.

[7] C. Y. Lee. An algorithm for path connections and its applications. *IRE Trans. Electronic Computers*, EC10(3):346–365, Sept. 1961.

[8] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. M. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Puclishers, Boston, 1984.

[9] O. Coudert and J. C. Madre. Implicit and incremental computation of primes and essential primes of boolean functions. In *in Proc. ACM/IEEE Design Automation Conf.*, pages 36–39, June 1992.