Integration of Logic Synthesis and Layout Processes by Generating Multiple Choices of Circuit Transformation

Hiroaki Yoshida Motohiro Sera Masao Kubo Masahiro Fujita

Department of Electronics Engineering
University of Tokyo

Abstract

In this paper, we propose a new methodology to integrate multiple circuit tranformations and routing processes. More specifically, this paper shows ways to utilize multiple choices of circuit transformations in routing processes. First, we introduce a new logic representation that implements all possible wire reconnections implicitly by enhancing global flow optimization techniques. Then we present two approaches for performing routing and wire reconnection simultaneously: exact approach and practical approach with commercial P/R tools. Since our methods take into account multiple circuit transformations during routing phase where the accurate physical information is available, we can obtain better results than the conventional routing tools. In addition, we can succeed in routing even if other routers like rip-up and reroute methods fail. We built a prototype system that implements the methods and preliminary results are reported.

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.

Global flow optimization technique[1][2] can reconnect a target wire without changing any other wires. Therefore the technique can easily be applied to a circuit even after its placement is fixed.

In this paper, we first introduce a new logic representation that implements all possible wire reconnections implicitly. Then we present two approaches for performing routing and wire reconnection simultaneously. Since our methods take into account multiple circuit transformations during routing phase where the accurate physical information is available, we can obtain better results than the conventional routing tools.

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 that we can derive all solutions from the representation. Section 4 describes our approaches to unify routing and wire reconnection. Experimental results are presented in Section 5. In Section 6, we describe the extention of our methodology to handle general circuits.

2. Overview of Existing Algorithms

2.1. Global Flow Optimization

In [1], global flow optimization technique has been proposed. It gathers circuit informations 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[2]. They have also introduced a new graph 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. Next, an implication flow graph is constructed as follows.

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

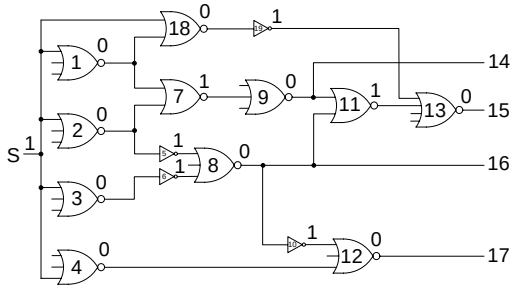

Figure 1. Example circuit.

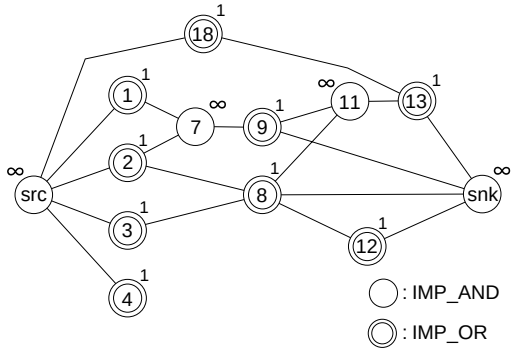

Figure 2. Implication flow graph.

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. 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 [3], Schmiedle *et al.* have presented an exact approach for solving channel routing problems. They use Multi-valued Decision Diagrams(MDDs) for representation of the routing space. All possible solutions are represented in a single MDD. Later, they have shown a search space reduction technique[4].

We briefly review the exact routing method. The method assumes that the grid-based model is used, *i.e.*, the routing 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 $\{1, ..., n\}$. $m_{xyz} = k$ 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 $C^0_{i,t}(p) = (p = i)$ 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. An MDD n_i representing all possible paths of net N_i can be derived from the equation:

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

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.

$$s = \bigwedge_{N_i \in N} n_i \qquad (3)$$

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.

For example, suppose new variables are $t_1, ..., t_n$. Then the resulting Boolean function is represented by $f(t_1, ..., t_n)$. Figure 3 shows the logic representation of the implication flow graph in Figure 2.

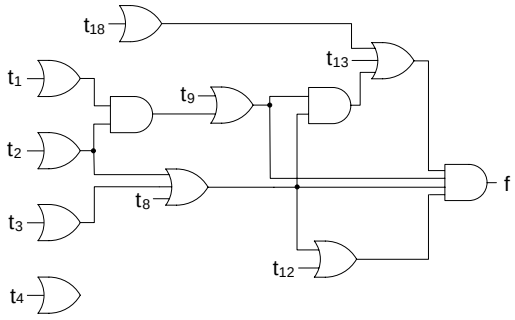

Figure 3. Logic representation of the implication flow graph.

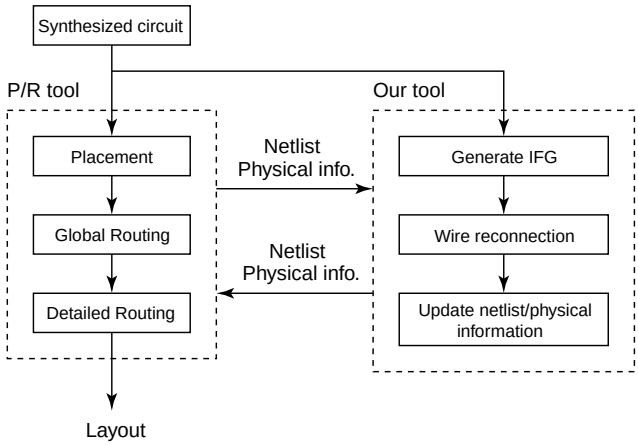

Figure 4. Proposed flow of the practical approach.

4. Integration of Wire Reconnection and Routing

4.1. Exact Approach

In this sub-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. We assume that candidate terminals for reconnection are already placed in a routing region and of course the locations of them are known. 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:

$$n_i = f(C_{i,t_{i1}}(t_{i2}), ..., C_{i,t_{i1}}(t_{in})) \quad (4)$$

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 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. 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 3, the function f is minimized to:

$$f = t_1 \cdot t_2 + t_2 \cdot t_9 + t_3 \cdot t_9 + t_8 \cdot t_9 \quad (5)$$

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.

As shown in the original paper[3], 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, the complexity of the problem becomes smaller. We just 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.

4.2. Practical Approach with Commercial P/R Tools

In the previous sub-section, an exact approach for solving the reconnection and routing problems simultaneously has been proposed. Although the approach can solve the reconnection and routing problems exactly and simultaneously, it cannot handle large problems. For large problems, we propose another approach. The proposed flow is illustrated in Figure 4.

In this flow of the integration, when writing out new placement/routing information, the decision on which portion can be eliminated and which portion can be left is the most important and difficult issue.

In this paper, we treat a combinational circuit, and so no latch exists in a circuit although an extension to sequencial circuits is straightforward. A netlist written out consists of one module, and only the definition of input, output, and the instance of each cell are described in the module.

After making circuit transformation by the logic representation of implication flow graph explained in Section 3, cells are classified into three types. We define these three types of cells as follows.

Definition 4.1 *The cell corresponding to the node which does not change before and after circuit transformation is defined as "unchanged cell".*

Definition 4.2 *The cell corresponding to the node whose number of inputs decreses by one before and after circuit transformation is defined as "pruned cell". Especially, the*

cell which is deleted after circuit transformation is defined as "deleted cell".

Definition 4.3 *The cell corresponding to the node whose number of inputs increases by one before and after circuit transformation is defined as "connected cell".*

We explain the algorithm of writing new placement information below.

1. In the case of *unchanged cell*, we can just write out as it is since the placement information on the cell does not change.

2. In the case of *pruned cell*, we can also write out as it is since the width of the cell only decreases. In the case of *deleted cell*, we don't write out it to new placement information since the cell is lost.

3. In the case of *connected cell*, the width of the cell increases, and may overlap with right or left cell or interfere with the wiring space. Therefore, when there is a *connected cell*, we find the nearest *pruned cell* in the same row, and shift all cells between the cells. The decrease in width of a *pruned cell* is equal to the increase in width of *connected cell*, in the case of our technology. Therefore, when all rows satisfy that the number of *connected cells* is less than or equal to the number of *pruned cells*, it is called that the choice of the circuit transformation is *legal*. The algorithm selects the *legal* choice within the choices given by the logic representation of implication flow graph.

Routing information also change in the case of *pruned cell* and *connected cell* since the width of a cell changes. Moreover, in even *unchanged cell*, routing information also changes on the cells which are shifted in order to compensate for the width which is increased by a *connected cell*. Therefore, we delete the routing information on all wires and contacts which are connected to the input and output pins of these cells. By this algorithm, we can safely leave the unchanged routing information to the maximum.

5. Experimental Results

5.1. New Logic Representation of Implication Flow Graph

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. The results are shown in Table 1. The fourth column shows the number of the nodes with ten or more fanouts. In column 5 and 6, the number of wire reconnections for the nodes having ten or more fanouts and the average of them are shown respectively.

Table 1. Experimental results of new logic representation of implication flow graph.

circuit	*#lits*	*#nodes*		*#choices*	*ave.*	*CPU*
		all	≥ 10			*(sec)*
lal.blif	300	192	7	253	36	0.2
c8.blif	370	233	6	634	106	0.2
ttt2.blif	486	260	15	316	21	0.3
alu2.blif	747	376	9	7	0.8	0.6
x1.blif	747	428	12	788	66	0.4
x4.blif	978	608	18	26	1.4	0.5
rot.blif	1288	903	6	4	0.7	0.6
x3.blif	1776	997	27	34	1.3	1.0
pair.blif	3258	1972	2	434	217	4.4
all			102	2492	24	

5.2. Exact Approach

The algorithm presented in Section 4.1 and [3] was implemented and we performed experiments on some example circuits. We consider an example circuit that consists of three nets and routing grids with a channel that has 6 columns, 4 tracks and 2 layers as shown in Figure 5. The Boolean functions that are derived from the improved global flow algorithm are as follows:

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

In Equation (6)-(8), the Boolean functions f_x, f_y, f_z correspond to the terminals x, y, z in Figure 5 respectively and each Boolean variable corresponds to the terminal in Figure 5. That is, the terminal x can be connected to the terminals c, e, g or the terminals d, e, g. The terminal y must be connected to terminals b, f, h and the terminal z to the terminals a, i. We were able to find all possible solutions within 800 seconds. Figure 5 shows one of the solutions. For larger example, we couldn't obtain any results because the MDDs blew up.

Since MDDs are encoded to BDDs in our implementation, 96 Boolean variables are needed for the example. Generally, BDDs can handle up to 200 variables. Due to this, the algorithm can handle the problems with up to 200 variables, for example, the problem with 8 columns, 6 tracks, 2 layers and 3 nets.

For larger examples, the heuristic technique described in Section 4.1 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 iterations. In column 5 and 6, memory usage and CPU time are shown respectively.

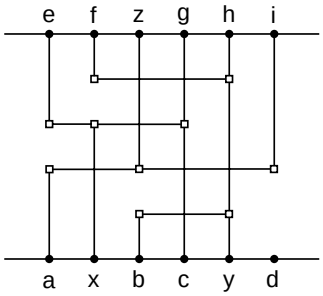

Figure 5. Example routed by exact approach.

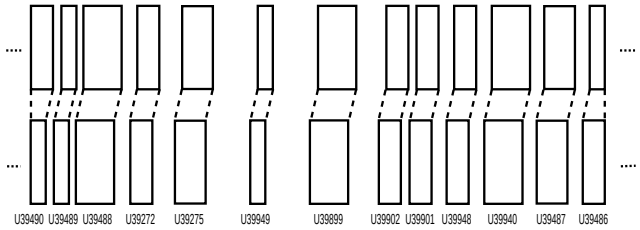

Figure 6. Cell placements before and after circuit transformation.

Table 2. Experimental results of exact approach with a 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

5.3. Practical Approach with Commercial P/R Tools

We built a prototype system which implements the flow shown in Figure 4 and performed an experiments on MCNC benchmark circuit x3.blif which consists of 763 NOR gates. The design was mapped on our three metal layer 0.35*um* standard cell library using Synopsys® Design Compliler™, and placed and routed by Avant!® Apollo™ initially. Wire reconnections were done as follows.

1. After placement and routing, one net was found to be unrouted. The net had 84 fanouts.
2. The logic representation of the implication flow graph was generated and 8 choices of wire reconnections were found. Only one of them is legal.
3. After a wire reconnection, re-placement and re-routing were performed without any errors. 0 out of 763 cells are re-placed and 37 out of 936 nets are re-routed.

Cell placements for a row of the standard cell layout before and after wire reconnection are shown in Figure 6. In the figure, the top row is the placement before reconnection and the bottom is the placement after reconnection. The leftest cell is a *pruned cell*, while the rightest cell is a *connected cell*.

6. Extension for General Circuits

Since the methodology described above assumes that circuits consist of only NOR gates, it cannot handle general circuits containing complex gates such as XOR, AND-OR-INVERTER, *etc*. When we handle such complex gates, we first decompose them into the NOR gates and apply the global flow optimization technique. If a candidate wire for reconnection is the wire which was originally inside the complex gate, we decompose the original gate or give it up. The decomposition causes the needs of incremental re-mapping and re-placement. Generally, incremental re-mapping and re-placement are both complicated problems. However, we can choose one of wire reconnections to make incremental re-mapping and re-placement easier since we have multiple choices of wire reconnections.

7. Conclusions and Future Works

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 previous routing methods. We also presented a practical approach with commercial P/R tools. It is based on current design flow and can reduce design iterations significantly. The priminary experiments show that our methodology works on the practical design flow. We are now preparing for more experiments.

References

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

[2] S. C. Chang, Z. Z. Wu, and H. Z. Yu. Wire reconnections based on implication flow graph. In *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 2000.

[3] F. Schmiedle, R. Drechsler, and B. Becker. Exact channel routing using symbolic representation. In *Proc. IEEE Int. Symp. Circuit and Systems*, May 1999.

[4] F. Schmiedle, D. Unruh, and B. Becker. Exact switchbox routing with search space reduction. In *Proc. ACM Int. Symp. Physical Design*, Apr. 2000.