A Redesign Technique for Combinational Circuits Based on Gate Reconnections

Yuji Kukimoto
University of California
Berkeley, CA 94720

Masahiro Fujita
Fujitsu Laboratories of America
San Jose, CA 95134

Robert K. Brayton
University of California
Berkeley, CA 94720

Abstract

In this paper, we consider a redesign technique applicable to combinational circuits implemented with gate-array or standard-cell technology, where we rectify an existing circuit only by reconnecting gates on the circuit with all the gate types unchanged. This constraint allows us to reuse the original placement as is, thereby speeding up the total time needed for a redesign. We formulate this problem as a Boolean-constraint problem and give a BDD-based algorithm to check the feasibility of redesign.

1 Introduction

Incremental synthesis is a synthesis technique which reuses existing circuits to come up with circuits satisfying new specifications. Since engineering changes arise frequently in actual design processes, the technique is of practical importance from an industrial point of view. Several synthesis techniques have already been proposed for combinational circuits [2, 3, 9, 8] and sequential circuits [1], where an additional logic is attached before and/or after an existing circuit or a part of the existing circuit is replaced with a new logic so that the final circuit follows a new specification. In [5], a redesign technique for lookup-table type FPGA's was proposed, where, given a combinational circuit mapped on an FPGA, the circuit is rectified by only changing the functionality of lookup-tables with all the routing preserved. This approach is natural for lookup-table type FPGA's since modifying the functionality of tables can be done by reprogramming the tables, which is cheap, whereas rerouting requires an expensive physical design all over again.

In this paper, we consider a redesign technique applicable to combinational circuits implemented with gate-array or standard-cell technology, where we rectify an existing circuit only by reconnecting gates on the circuit with all the gate types unchanged. Note that our constraint is symmetric with the one used in FPGA rectification [5]. This constraint is reasonable in our set-up since, in gate-array or standard-cell based designs, one can make a modification to routing without changing the placement of the gates as long as there is space left for the routing. This way we get around an expensive physical placement required otherwise. Furthermore in standard-cell designs this enables us to reuse masks for the original placement. Note that in industries this type of rectification is actually done manually by designers. We formulate this problem as a Boolean-constraint problem and give an algorithm based on BDD's, which helps designers determine the feasibility of redesign.

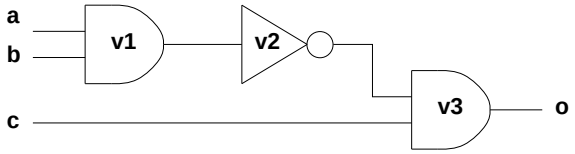

Figure 1: Example – The Original Circuit

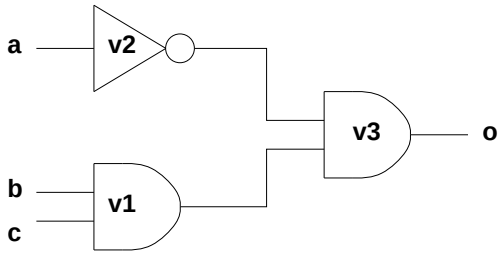

Figure 2: Example – A Redesigned Circuit

This paper is organized as follows. In Section 2, the proposed redesign technique is informally discussed with an example. Section 3 formulates the problem and gives an algorithm to solve it. Experimental results are shown in Section 4. Section 5 concludes the paper.

2 Example

In this section we present the basic idea of our approach using a simple example.

Suppose we have a circuit shown in Figure 1. This is a combinational circuit with two 2-input AND gates and a single inverter, whose functionality is $o = (\overline{a} + \overline{b})c$. Assume that we like to redesign this circuit only by reconnecting the gates so that its functionality is $o = \overline{a}bc$. Figure 2 shows a circuit for the new specification. Note that the circuit is composed of the same set of the gates in the original circuit and the only difference between the two is in the connections between the gates. More precisely, in the original circuit we have the following five connections: from

a to v_1, from b to v_1, from v_1 to v_2, from v_2 to v_3 and from c to v_3, whereas the redesigned circuit has different connections, i.e. from a to v_2, from b to v_1, from c to v_1, from v_1 to v_3 and from v_2 to v_3.

Since this transformation only involves changes in connections, the modifications required for the layout of the circuit are restricted to its routing portion, i.e. the placement information can be reused completely. It is true that an extremely congested layout may not allow reconnections of two points far away on the layout under the current placement, but by restricting our focus on physically feasible reconnections involving adjacent gates, our claim is validated. This can be done automatically by investigating the original layout.

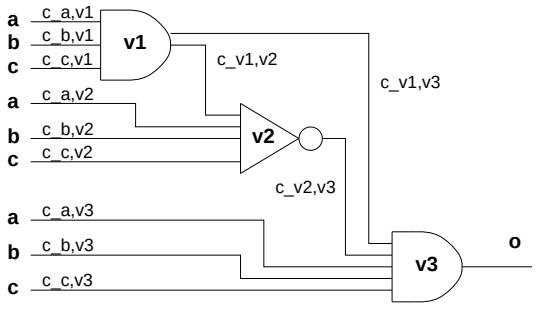

Figure 3: Connection Variables

3 Rectification Algorithm

In this section we formulate the rectification problem described in the previous section as solving a set of Boolean constraints, for which an algorithm based on BDD's is given. Remember that we are working on redesigns of combinational circuits. The key idea behind this approach is to introduce a Boolean variable, called *a connection variable*[1], for each ordered pair of gates to represent if there exists a connection from the output of the first gate to an input of the second gate in a redesigned circuit. In other words, the variable takes 1 if we have the connection, and 0 otherwise. This way our goal is now to compute a 0-1 assignment to the variables which makes the reconnected circuit behave exactly the same as a new specification. The first step of this algorithm is to compute the functionality of all the reconnected circuits in terms of primary inputs, primary outputs and connection variables. Note that each 0-1 assignment to the connection variables corresponds to a single reconnected circuit. Therefore, this function captures all the functionality which can be realized by our redesign scheme. Having computed this, we proceed to compare this functionality with a new specification to extract the constraint which has to be satisfied for the rectification. The final constraint is represented as a characteristic function in terms of connection variables. If the function is equal to 0, then we know there is no solution. Otherwise, a satisfying assignment of the function is chosen based on a given criterion, i.e. the minimum number of connections or the minimum change from the original circuit. The detailed algorithm is given in the following, where we assume that every Boolean operation is performed with BDD's.

3.1 Connection Variables

In general we should introduce a connection variable for every ordered pair of gates so that any connection is considered. Under this assumption, a redesigned circuit could have a combinational loop, i.e. a loop without a latch on it. The circuit can be considered valid if there is no oscillation at primary outputs, but in this paper we restrict ourselves to finding a circuit without any combinational loop. For this purpose, connection variables are introduced such that no combinational loop is obtained under any 0-1 assignment to the variables. A simple way to guarantee this condition is to give a linear order to the set of gates and introduce connection variables only for the increasing direction in this order[2]. Currently we extract the linear order from the original network with a depth-first traversal so that the structural similarity between the original network and a redesigned network is observed. Note that we introduce a connection variable for any connection from a primary input to a gate since this never produces a combinational loop.

For example, we introduce the following twelve connection variables for the circuit in Figure 1 under a linear order $v_1 < v_2 < v_3$: $c_{a,v_1}, c_{b,v_1}, c_{c,v_1}, c_{a,v_2}, c_{b,v_2}, c_{c,v_2}, c_{v_1,v_2}, c_{a,v_3}, c_{b,v_3}, c_{c,v_3}, c_{v_1,v_3}, c_{v_2,v_3}$, where $c_{s,t}$ is the variable for the connection between the output of gate s and an input of gate t. Notice that no assignment to the connection variables makes the reconnected circuit cyclic[3]. For the correspondence between connections and their associated variables, see Figure 3.

3.2 Computing the Set of All Functionality Realized by Reconnections

Once a set of connection variables is fixed, the next step is to compute the set of all functionality that can be realized by reconnections. We represent this implicitly as a characteristic function of primary inputs, primary outputs, and connection variables. Each 0-1 substitution to connection variables in this characteristic function yields the characteristic function which represents the input-output behavior of the corresponding reconnected circuit.

We first give a technique to model the functionality of each gate with a characteristic function and then discuss how to build up the final characteristic function from the set of component characteristic functions obtained from all the gates.

Let us assume that we have a two-input AND gate v_k in the original network. We are interested in the functionality of this

[1] The same idea was used by Muroga *et al.*[7] in the formulation of multi-level logic synthesis based on integer programming.

[2] Another possible way is to partition the set of gates into several sets, give an integer level to each set, and introduce a connection variable only for a connection from a gate in a lower set to a gate in an upper set.

[3] As a more general set-up, one can introduce a connection variable for every ordered pair of gates and compute, with Boolean-function matrix manipulation, the condition that the reconnected circuit has a loop [4], which gives us a systematic way to remove all cyclic circuits from our consideration. This general approach, however, is likely to cause a BDD blowup in the following stages due to the absence of a good variable ordering.

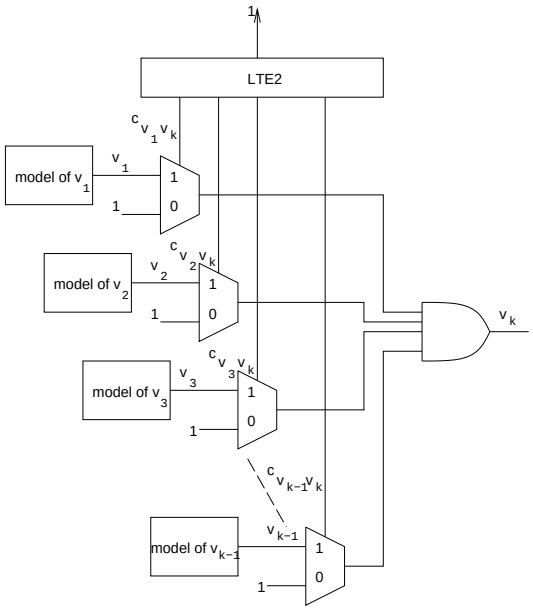

Figure 4: Model of the 2-input AND gate

gate when reconnection is allowed. Let $c_{v_i,v_k}(i = 1, \ldots, k-1)$ be the Boolean variable for the connection from the output of gate v_i to an input of the AND gate v_k. Let v_i be the Boolean variable corresponding to the output of gate v_i. Then this gate can be modeled as the circuit extended with selectors shown in Figure 4. Each selector connects the output of gate v_i to an input of the AND gate iff the corresponding connection variable c_{v_i,v_k} is set to one. Otherwise, the non-controlling value of the AND gate, a Boolean value one, is fed into the input. In addition, since the original AND gate has two inputs, at most two connection variables out of $c_{v_1,v_k}, \ldots, c_{v_{k-1},v_k}$ can evaluate to one[4].

Thus, the characteristic function of the circuit in Figure 4, $\chi_{\text{2-AND}}$, is:

$$\chi_{\text{2-AND}} = \text{LTE2}(c_{v_1,v_k}, \ldots, c_{v_{k-1},v_k}) \cdot (v_k \equiv \prod_{i=1}^{k-1}(c_{v_i,v_k} \Rightarrow v_i))$$

where LTE2() is the Boolean function which evaluates to one iff less than or equal to two arguments are one. Note that LTEn() is easily computed recursively. (See details for Appendix.) Furthermore, LTEn() is a symmetric function, thus the size of the BDD is guaranteed to be small independent of variable orderings.

One can easily see that a similar model exists for any type of gates[5]. For example, the characteristic functions of a 2-input OR gate, a 2-input XOR gate and an inverter are defined as follows.

$$\chi_{\text{2-OR}} = \text{LTE2}(c_{v_1,v_k}, \ldots, c_{v_{k-1},v_k}) \cdot (v_k \equiv \sum_{i=1}^{k-1}(c_{v_i,v_k} \cdot v_i))$$

$$\chi_{\text{2-XOR}} = \text{LTE2}(c_{v_1,v_k}, \ldots, c_{v_{k-1},v_k}) \cdot (v_k \equiv \bigoplus_{i=1}^{k-1}(c_{v_i,v_k} \cdot v_i))$$

$$\chi_{\text{INV}} = \text{LTE1}(c_{v_1,v_k}, \ldots, c_{v_{k-1},v_k}) \cdot (v_k \equiv \sum_{i=1}^{k-1}(c_{v_i,v_k} \cdot \overline{v_i}))$$

Having defined the characteristic function of each gate type, one can construct the characteristic function of the whole circuit by ANDing the component characteristic functions of all the gates since all the input-output constraints have to be satisfied simultaneously in the final circuit. Formally, the characteristic function of the circuit after reconnection, χ , is computed as follows.

$$\chi(\mathbf{v}, \mathbf{c}) = \prod_{g \in G} \chi_g(\mathbf{v}, \mathbf{c})$$

where G is the set of all the gates in the original network and χ_g is the component characteristic function for gate g. Note that **v** is the set of variables containing all the primary inputs **i**, all the primary outputs **o**, and all the internal variables **t** associated with intermediate nodes[6] and **c** is the set of all the connection variables. Thus, one can see $\chi(\mathbf{v}, \mathbf{c})$ as $\chi(\mathbf{i}, \mathbf{o}, \mathbf{t}, \mathbf{c})$, which describes the constraint among primary inputs, primary outputs, internal variables and connection variables imposed by the structure of the circuit.

Now, since we are only interested in the input-output behaviors of the circuit under various assignments to connection variables, internal variables can be dropped off from χ by the smoothing operation.

$$\chi_D(\mathbf{i}, \mathbf{o}, \mathbf{c}) = \mathcal{S}_{\mathbf{t}}\chi(\mathbf{i}, \mathbf{o}, \mathbf{t}, \mathbf{c})$$

This computation can be sped up by merging the AND operations in $\chi(\mathbf{i}, \mathbf{o}, \mathbf{t}, \mathbf{c})$ with the smoothing operation by the `and_smooth` operation. More precisely, the component characteristic functions of $\chi(\mathbf{i}, \mathbf{o}, \mathbf{t}, \mathbf{c})$ are AND-ed from primary outputs to primary inputs in a reverse topological order with simultaneously smoothing out variables in **t**. Furthermore, each LTE term in a component characteristic function can be pulled out of the smoothing operation since it doesn't depend on **t**.

Let us follow the above construction with the example in Section 2. The characteristic functions of the three gates in the circuit are:

$$\begin{aligned}\chi_{v_1} \quad = \quad & \text{LTE2}(c_{a,v_1}, c_{b,v_1}, c_{c,v_1}) \\ & \cdot [v_1 \equiv (c_{a,v_1} \Rightarrow a)(c_{b,v_1} \Rightarrow b)(c_{c,v_1} \Rightarrow c)]\end{aligned}$$

[4]Note that it is valid that only a single connection variable takes one since this means that the AND gate is used as a buffer by feeding a Boolean one to the remaining input. Similarly all the connection variables can be set to zero, in which case this gate is not used in the redesigned circuit.

[5]A slightly elaborate model is needed for complex gates especially when the gates are not symmetric, in which case we have to introduce more than one connection variable for an ordered pair of gates to distinguish inequivalent inputs.

[6]Here we assume that for each primary output the same gate which feeds into the output in the original network is used for feeding the output. In other words, we don't change any connections directly connected to primary outputs. We can formulate a more general problem where this constraint is absent, by introducing for each primary output a buffer, whose input is selected from the output of the gates in the network. The same modeling technique using connection variables works in this case. One has only to multiply this new constraint to χ.

$$
\begin{aligned}
\chi_{v_2} &= \mathrm{LTE1}(c_{a,v_2}, c_{b,v_2}, c_{c,v_2}, c_{v_1,v_2}) \\
&\quad \cdot [v_2 \equiv c_{a,v_2}\overline{a} + c_{b,v_2}\overline{b} + c_{c,v_2}\overline{c} + c_{v_1,v_2}\overline{v_1}] \\
\chi_{v_3} &= \mathrm{LTE2}(c_{a,v_3}, c_{b,v_3}, c_{c,v_3}, c_{v_1,v_2}, c_{v_2,v_3}) \\
&\quad \cdot [v_3 \equiv (c_{a,v_3} \Rightarrow a)(c_{b,v_3} \Rightarrow b)(c_{c,v_3} \Rightarrow c) \\
&\qquad (c_{v_1,v_3} \Rightarrow v_1)(c_{v_2,v_3} \Rightarrow v_2)]
\end{aligned}
$$

Now the characteristic function of a set of reconnected circuits, χ_D, is defined as:

$$\chi_D(a, b, c, o, c_{a,v_1}, \ldots, c_{v_2,v_3}) = \mathcal{S}_{v_1,v_2,v_3}\chi_{v_1}\chi_{v_2}\chi_{v_3}(o \equiv v_3)$$

3.3 Extracting Constraints on Connection Variables

At this point we are ready to compare the functionality represented by this characteristic function $\chi_D(\mathbf{i}, \mathbf{o}, \mathbf{c})$ with the characteristic function of a new specification, $\chi_S(\mathbf{i}, \mathbf{o})$. Note that one can easily compute the characteristic function of a specification given either as a completely specified function, an incompletely specified function or a Boolean relation. The condition on the connection variables **c** is that the input-output behavior of a reconnected circuit is included in the input-output behavior of the specification, namely,

$$\chi_{\text{redesign}}(\mathbf{c}) = \mathcal{C}_{\mathbf{i},\mathbf{O}}(\chi_D(\mathbf{i}, \mathbf{o}, \mathbf{c}) \Rightarrow \chi_S(\mathbf{i}, \mathbf{o}))^7$$

The consensus operator in the above extracts all the 0-1 assignments to **c** such that $(\chi_D(\mathbf{i}, \mathbf{o}, \mathbf{c}) \Rightarrow \chi_S(\mathbf{i}, \mathbf{o}))$ gets equal to a tautology, which means that the reconnected circuits corresponding to the 0-1 assignments follow the specification. Therefore, we first check to see if χ_{redesign} is a negative tautology. If that is the case, then no solution exists under our redesign scenario. Otherwise, any satisfying assignment to **c** gives a solution.

Let us go back to the example. Remember that our new specification in the example is $o = \overline{a}bc$. Therefore, the characteristic function of the specification, χ_S, is:

$$\chi_S(a, b, c, o) = [o \equiv \overline{a}bc]$$

Combined with χ_D, the final constraint on connection variables, χ_{redesign}, is computed as follows.

$$
\begin{aligned}
&\chi_{\text{redesign}}(c_{a,v_1}, \ldots, c_{v_2,v_3}) \\
&\quad = \mathcal{C}_{a,b,c,o}(\chi_D \Rightarrow \chi_S) \\
&\quad = \overline{c}_{a,v_1}c_{b,v_1}c_{c,v_1}c_{a,v_2}\overline{c}_{b,v_2}\overline{c}_{c,v_2}\overline{c}_{v_1,v_2}\overline{c}_{a,v_3}\overline{c}_{b,v_3}\overline{c}_{c,v_3}c_{v_1,v_3}c_{v_2,v_3}
\end{aligned}
$$

For this particular example, the characteristic function has a single minterm, which means that there is a single solution for this problem. Note that the reconnections implied by the minterm yields the circuit shown in Figure 2.

In general, however, since the final characteristic function has more than one minterm, we are interested in a solution optimum in some sense. For simplicity, we assume in the following that any connection requires the same amount of routing resource[8]. Under this assumption, two possible criteria are:

1. minimum number of connections
2. minimum number of reconnections which transform the original circuit to a redesigned circuit

The first criterion gives us a new circuit which requires the minimum routing resource, which corresponds to the satisfying assignment of χ_{redesign} that has the least number of 1's. Given χ_{redesign} in a BDD, this solution can be computed in time linear to the size of the graph based on the Lin-Somenzi procedure[6]. The second criterion yields a solution which requires the minimum change in the original circuit. The corresponding assignment to **c** is the satisfying assignment of χ_{redesign} that has the minimum Hamming distance from the assignment $\tilde{\mathbf{c}}$ which corresponds to the original circuit. Note that the original circuit is also characterized by some assignment to **c**, which we call $\tilde{\mathbf{c}}$ here. Figure 5 shows an algorithm to compute the minimum Hamming distance between satisfying assignments of f and a minterm x, given f as a BDD. The first two lines are terminal conditions. If $f = 1$, then x itself is guaranteed to be a satisfying assignment of f. Therefore, the minimum Hamming distance is 0. If $f = 0$, however, no satisfying assignment exists for f. Thus the procedure returns infinity. If the terminal conditions are not met, then we first look into the cache which maintains the previous results and return the corresponding result if we have it. Otherwise, we proceed to

```
    minimum_hamming_distance(f, x)
1       if ( f = 1 ) return 0
2       if ( f = 0 ) return ∞
3       if ( ∃(f, d) in cache ) return d
4       c is the top variable of f
5       d0 = minimum_hamming_distance(f_c̄, x)
6       d1 = minimum_hamming_distance(f_c, x)
7       if ( x_c ≠ 0 )
8           d = min(d0 + 1, d1)
9       else
10          d = min(d0, d1 + 1)
11      if ( min is achieved by the first argument above )
12          f.link = f_c̄
13      if ( min is achieved by the second argument above )
14          f.link = f_c
15      save (f, d) in cache
16      return d
```

Figure 5: Algorithm to Compute the Minimum Hamming Distance

[7] Unless the specification is given as a Boolean relation, this computation can be transformed into a simpler one by processing each output separately and multiplying each constraint to construct $\chi_{\text{redesign}}(\mathbf{c})$.

[8] One can evaluate a routing cost in a more reasonable way by assigning a weight to each connection depending on the difficulty in routing the connection, for example using the distance of two terminals of the connection in the layout. Note, however, that this extension can be easily handled in the algorithms presented afterwards without changing their time-complexity.

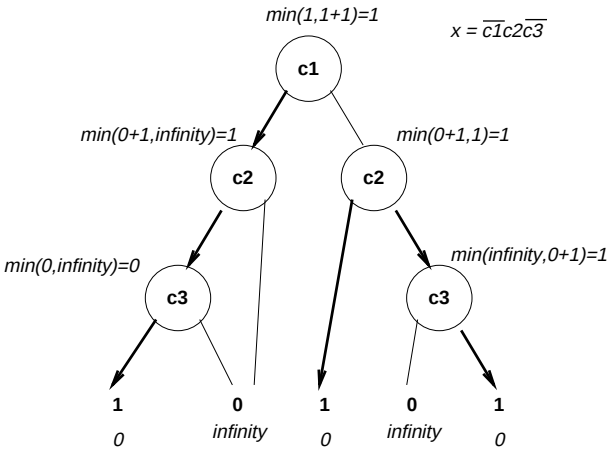

Figure 6: Computing the Minimum Hamming Distance on BDD

pick a splitting variable c as the top variable of f, solve two subproblems separately, and construct the solution from the solutions of the two. Two different cases have to be distinguished in this process. The first case is when x lies in the half space c(line 7), in which case the minimum of $d_0 + 1$ and d_1 is returned, where d_0 and d_1 are the solutions of the two subproblems for the negative and the positive cofactors with respect to c respectively. Note that we have to use $d_0 + 1$ instead of d_0 since setting c to 0 increases the Hamming distance by one in the case where x has the opposite value, i.e. 1, in c-coordinate. If not, then the procedure returns the minimum of d_0 and $d_1 + 1$ based on the symmetric argument[9]. The procedure runs in time linear to the size of a given graph. Each BDD node has an additional entry called *link*, which points to the child node which achieves the minimum value. Thus, by traversing the BDD along with this link information after calling `minimum_hamming_distance`, the minterm of f which achieves the minimum distance can be extracted in time linear to the number of supports of f.

Let us take an example to show how the procedure works. Let $f = c_1c_3 + \bar{c}_2\bar{c}_3$ and $x = \bar{c}_1c_2\bar{c}_3$. Figure 6 shows how the minimum Hamming distance is computed on the BDD of f. An Italic number attached to a node denotes the minimum Hamming distance between x and the minterms of the function represented by the node. An arrow on a BDD edge is a link connected by the procedure. A traversal of the BDD from the root to a 1-leaf along with links gives the satisfying assignment of f which achieves the minimum distance one, i.e. $\bar{c}_1\bar{c}_2\bar{c}_3$.

[9]If the top variable of f_c is not the next variable of c, then for the skipped variables between the two we can always choose a 0-1 assignment so that it exactly matches x. Therefore, these variables don't contribute anything to the minimum distance.

4 Experimental Results

We have implemented the proposed redesign algorithm on top of SIS using BDD's[10]. Table 1 shows preliminary experimental results of our rectification method on DEC 5900/260. `ex1` is an industrial example encountered in an engineering change of an embedded micro-processor. It is a part of the circuit extracted by a designer, which was known from a manual redesign to be rectifiable under the assumption that only reconnections are permissible. `ex2` is a simplified version of `ex1`. In both of these examples the algorithm successfully finds solutions. Although the circuits are small, the CPU time needed for this computation is fairly large. This is due to a large number of connection variables introduced for the formulation. One way to get around this is to extract constraints on reconnections from the layout and utilize that information to reduce the number of connection variables. For example, some reconnection might not be routable on the current placement since it is blocked somewhere in the middle or the two terminals are simply located too far, which implies that no connection variable is needed for this connection. Following this idea, we conducted further experiments where only a subset of the connection variables is introduced. Table 2 shows the relationship between the number of connection variables and CPU time. Note that the reduction of variables significantly improves CPU time. We expect that the technique is practical also for larger circuits(e.g. circuits composed of 20 gates) if the number of connection variables is controlled by the layout information.

5 Conclusions

We have discussed a redesign scheme based on gate reconnections, which is applicable to technology-mapped combinational circuits. This technique allows us to rectify a given circuit with the same set of gates in the original circuit. This fact is of significant importance in the case where the circuit is already laid out since the rectification is now completed by only changing the routing portion of the layout without redoing the whole layout. We have formulated the problem as a Boolean constraint and have given an algorithm to solve it. Experimental results showed that only small-sized problems are tractable due to the inherent complexity of the problem in the most general setting where any pair of gates is connectable, but additional constraints derived from layouts decrease the complexity significantly. We are currently working on the further improvement of the algorithm with the efficient use of these constraints. In addition, to handle larger circuits, we are considering automatic extraction of promising subnetworks, to which the proposed technique is applied, using don't care information.

Acknowledgments

The authors would like to thank Xudong Zhao of CMU for his collaboration in the initial stage of this work.

[10]Dynamic reordering is set during the computation due to the difficulty in finding a good variable ordering.

circuit	# inputs	# outputs	# gates	# connection variables	CPU time(sec)
ex1	6	2	9	91	9830.5
ex2	6	2	7	64	91.3

Table 1: Experimental Results

circuit	# connection variables	CPU time(sec)
ex1	91	9830.5
ex1	63	2170.1
ex1	54	825.4
ex1	48	318.4
ex1	44	45.1

Table 2: Experimental Results — # Connection Variables and CPU Time

References

[1] M. Fujita. A method for automatic design error correction in sequential circuits. In *Proceedings of the European Conference on Design Automation(EDAC-93)*, pages 76–80, February 1993.

[2] M. Fujita, T. Kakuda, and Y. Matsunaga. Redesign and automatic error correction of combinational circuits. In *Proceedings of the IFIP TC10/WG10.5 Workshop on Logic and Architecture Synthesis*, pages 253–262. North Holland, May 1990.

[3] M. Fujita, Y. Tamiya, Y. Kukimoto, and K.-C. Chen. Application of Boolean unification to combinational logic synthesis. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 510–513, November 1991.

[4] N. Ishiura. Private communication, April 1994.

[5] Y. Kukimoto and M. Fujita. Rectification method for lookup-table type FPGA's. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 54–61, November 1992.

[6] B. Lin and F. Somenzi. Minimization of symbolic relations. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 88–91, November 1990.

[7] S. Muroga and T. Ibaraki. Design of optimal switching networks by integer programming. *IEEE Transactions on Computers*, C-21:573–582, June 1972.

[8] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney. The Transduction method – design of logic network based on permissible functions. *IEEE Transactions on Computers*, 38(10):1404–1424, October 1989.

[9] Y. Watanabe and R. K. Brayton. Incremental synthesis for engineering changes. In *Proceedings of IEEE International Conference on Computer Design*, pages 40–43, October 1991.

LTE$n(v)$
1 return LTE(v, m, n)

LTE(v, k, l)
1 if ($l < 0$) return 0
2 if ($k \leq l$) return 1
3 if ($\exists(v, k, l, f)$ in cache) return f
4 $f_0 =$ LTE$(v, k-1, l)$
5 $f_1 =$ LTE$(v, k-1, l-1)$
6 $f = \overline{v[k]} f_0 + v[k] f_1$
7 save (v, k, l, f) in cache
8 return f

Figure 7: Algorithm to Compute LTEn()

A An Algorithm to Compute LTEn()

Given a set of m variables, LTEn returns the function which evaluates to one if and only if less than or equal to n variables are set to one. Let $v[1 \ldots m]$ be the array which contains the m variables. Figure 7 shows a recursive algorithm to compute LTEn. Given a set of m variables in the array v, LTE$n(v)$ first calls a subprocedure LTE(v, m, n), which is then computed recursively. LTE(v, k, l) is designed so that it returns the function which evaluates to one iff less than or equal to l variables out of $v[1 \ldots k]$ are set to one. The first two lines of the algorithm are terminal cases. If $l < 0$, then since any 0-1 assignment to v has at least zero 1's the result is 0. If $k \leq l$, then any 0-1 assignment to v has at most k 1's, thus the procedure returns 1. If the result is not in the cache, then we split the problem into two subproblems. In one subproblem, by assuming that $v[k]$ is set to 0, we solve LTE$(v, k-1, l)$, while in the other problem LTE$(v, k-1, l-1)$ is computed assuming $v[k]$ gets 1. The two results are merged into the final solution in line 6.