Timing Optimization on Mapped Circuits

Ko Yoshikawa, Hiroshi Ichiryu, Hisato Tanishita,
Sigenobu Suzuki, Nobuyoshi Nomizu, Akira Kondoh*
NEC Corporation
1-10 Nisshin-cho, Fuchu-city, Tokyo, Japan

Abstract

This paper describes new techniques for timing optimization of CMOS or BiCMOS gate array or standard cell circuits. Based on previous works on critical path resynthsis, technology mapping algorithms using dynamic programming techniques, and fanout optimization algorithms, the following new techniques were developed: a hierarchical data structure in which a circuit is partitioned into subcircuits, a new weight function to grade subcircuits in terms of their potential for delay reduction, a critical path resynthesis technique preceded by non critical path resynthsis, a mapping algorithm using tree covering techniques tightly coupled with a fanout optimization algorithm which can treat dual signals not only in sinks but also in sources, and a correction procedure for short paths. A program using these techniques achieves an average speed up of 37% with 27% increase in area on the DAC'86 benchmark set plus several additional circuits from actual designs.

1 Introduction

Timing optimization of logic circuits is very important because high performance is one of the key factors in making logic devices competitive in the electronics industry. It becomes, however, very difficult or too time consuming for human designers to optimize large scale circuits on schematic diagrams within a limited amount of time. On the other hand, automatic optimization now becomes promising with advanced techniques which have been developed over the last decade.

In this paper, we describe several new timing optimization techniques for technology mapped circuits. The objective is to transform a given circuit so that timing constraints specified by the user, such as a clock cycle, are satisfied with minimum area increase. Our work is based on the critical path resynthesis on technology independent combinational circuits[1], the technology mapping algorithms using tree covering techniques [2-5], and the fanout optimization algorithms [5,6].

An algorithm in [1] extracts a subcircuit called the ϵ-network of critical paths in a technology independent combinational circuit, and then resynthesizes it to improve delay through the circuit. Although this technique is quite effective, it sometimes produces a circuit which has an unnecessary area increase and/or undesired performance when technology mapped, since delay estimation before mapping is not accurate enough. One possible solution is to devise a more accurate technology independent delay model [14]. Another solution we pursued is to apply this technique on a mapped circuit on which relatively accurate timing analysis can be carried out. To make this straightforward solution more effective, we developed the following new techniques: a hierarchical data structure in which a circuit is partitioned into subcircuits, a new weight function to grade subcircuits in terms of their potential for delay reduction, and a complementary method which improves delay through the outside of the ϵ-network so that the succeeding resynthesis within the ϵ-network can produce better results.

For remapping, we use a tree covering algorithm combined with a fanout optimization algorithm. The best gate for the root of a tree is selected from all feasible pairs of a gate and a fanout tree for both polarities at the root. This combination can be viewed as an extension of tree covering algorithms in which the load at the root is taken into account[4,5] so that the best gate at the root can be selected.

Although the fanout optimization problem on a realistic delay model has been shown to be NP-Complete by [7], several good heuristic algorithms have been proposed [5,6]. We propose a new algorithm which can handle both polarities in sinks as well as in sources. Utilizing both polarities in sources is important for circuits which include flip-flops or dual output gates. Our algorithm is a derivation of [8], called combinational merging, which is able to isolate critical paths. The original algorithm[7] assumed a simple delay model. An extended version to treat a realistic delay model can be found in Touati's comprehensive work on technology mapping and fanout optimization[9]. Our algorithm happens to be an enhancement of his approach. We adopt branch and bound techniques to explore a larger solution space.

Short paths correction must be done to assure correctly working synchronous circuits using various clocking scheme. We call it mimimum delay assurance. In LSS[10], local transformations are used to solve this problem. However, details are not given. We propose a procedure which consists of path tracing to look for good places to add delay and three types of local transformations.

In section 2, we introduce our timing optimization algorithms. Then, in section 3, we describe the results of the algorithms for a benchmark set and actual designs. Finally, in section 4, we describe our conclusions and future work.

2 The Algorithms

2.1 Overview

In this section, we describe the inputs, outputs, delay model and data structure used in our timing optimization program.

*NEC Engineering Corp.

The inputs are a technology mapped netlist of a circuit, timing constraints specified by the user, and a library. The input circuit may have been synthesized by the FUSION logic synthesis system [11] whose primary objective is to produce acceptable circuits, or may have been designed manually. The circuit can be combinational or synchronous using a single phase clock and edge triggered flip-flops. The timing constraints consist of a clock cycle, arrival times of primary inputs, required times of primary outputs, and estimated clock skew values for sets of flip-flops. The times can be specified in terms of rise/fall and maximum/minimum delays. The library includes delay information and function for each primitive block of the target technology, which can be CMOS or BiCMOS gate array, sea of gates, or standard cell. The output is also a technology dependent netlist of the circuit which is resynthesized to meet the timing constraints with minimum area increase.

For delay calculation, the block-oriented algorithm is adopted [12]. First, arrival times are calculated from primary inputs and flip-flop outputs. Next, required times are calculated from primary outputs and flip-flop inputs. In the latter case, we add a maximum clock skew value which is chosen from skews between the load flip-flop and all flip-flops which have paths to it. Slacks are calculated along with required times.

The delay model of each gate is represented as follows:

$$D_i k = I_i k + R_i * (C_r + \sum_h C_h) \qquad (1)$$

- $D_i k$ is the delay from the k-th input of gate g to the i-th output of gate g
- $I_i k$ is the intrinsic delay from the k-th input to the i-th output of gate g
- R_i is the resistance of the i-th output pin of gate g
- C_r is the estimated routing capacitance
- C_h is the input capacitance of load pin h

To estimate routing capacitance C_r, a simple wiring model which has an offset length and an additional length for each load is used. C_r is calculated as follows:

$$C_r = E * (L_o + n * L_i) \qquad (2)$$

where

- E is a coefficient
- L_o is the offset length
- n is the number of loads
- L_i is the additional length for each load

E, Lo and Li are specified by the user.

A circuit is represented as a DAG (directed acyclic graph) by cutting loops at flip-flops. A gate is represented as a node. A connection between gates is represented as an edge of the DAG. Besides this "gate level" DAG, another DAG where a node corresponds to a subcircuit, called a group, is constructed. In this latter DAG, four types of groups are defined.

(A) Random logic group

This type of group consists of gates included in a fanout free region of the circuit and its fanout tree. Figure 1 is an example of a random logic group. The root of both trees is called the root of the group. If a root gate is duplicated as in Figure 1 in the input circuit, it is checked and the duplicated gates are also included in the same group.

(B) Flip-flop group

One flip-flop (which is a root) and its fanout tree(s).

(C) Input group

A primary input (which is a root) and its fanout tree.

(D) Other group

A gate which has several different output functions and its fanout trees.

For example, a four bit adder gate and its fanout trees are classified as this type of group.

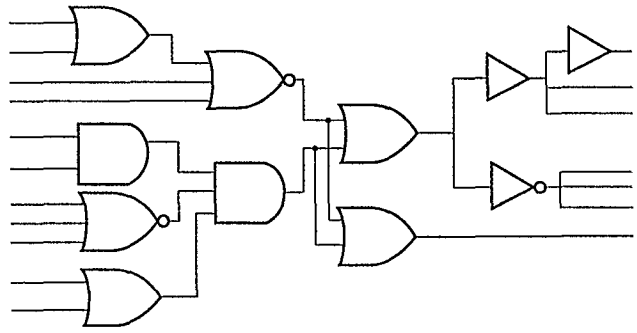

Figure 1 : An example of a group

```
Timing_optimization ( )
{
    delay_calculation ( ) ;
    while (the slack of critical path, < 0
                and it is possible to speed up ) {
        Sg = bottle neck_analysis ( ) ;
        if (Sg = ∅)    end ;
        foreach (G ∈ Sg) {
            resynthesis (G)
        }
        incremental_delay_calculation ( ) ;
    }
    area_recovery ( )
    minimum_dealy ( )
}
```

Figure 2 : Timing optimization loop

2.2 Timing Optimization Procedure

Figure 2 describes the overall process of timing optimization. After the initial delay calculation, the loop for delay optimization starts. The execution of this loop ends when the delay constraints are satisfied, or when there is no room for further optimization. In the loop, a set of groups which forms a cut set of the DAG is extracted. This extraction is called bottleneck analysis and a correlation coefficient is used as a cost to grade groups for delay optimization. Next, for each extracted group, resynthesis is executed to speed up the circuit. After all extracted groups are processed, delay recalculation is carried out incrementally.

After the loop, there may be groups whose timing constraints have been relaxed. Because these may be realized with less area, they are remapped in the subprocedure called *area_recovery*(). Finally, minimum delay is assured to correct for short paths.

2.3 Bottleneck Analysis

A set of groups which is feasible to improve maximum delay through the circuit is chosen by one of the following two methods. The first one selects groups from an ϵ-network[1] which corresponds to a subnetwork including the most critical paths or so-called "bottleneck" of the circuit in terms of delay. The other selects groups from outside the ϵ-network those can be viewed as side inputs and outputs of the bottleneck. In the first method, a network of groups whose root slacks are within ϵ from the most negative slack, called an ϵ-network, is extracted. Then, the weight of all groups in the ϵ-network is calculated as described in section 2.4. Finally, a minimum weighted cutset of the ϵ-network is formed.

The method is based on the one used in mis[1], with three differences. First, their method works on a gate level DAG while ours works on a group level DAG. Since the number of nodes is smaller in a group level DAG, smaller computation time is required in general. Second, a different weight function is used to assign a weight to each node in the DAG. Third, theirs is for a technology independent network while ours is for a mapped network on which more accurate timing calculation can be carried out.

The second method chooses groups from the outside of the ϵ-network. Improving the delay through the outside has complementary effects on improving the delay through an ϵ-network. For example, suppose there is a 3 input AND gate in an ϵ-network which has one input from the inside and the other inputs from the outside of the ϵ-network. Now, assume that the slacks of the outside inputs are just slightly larger than the slack of inside input. In this case, a decomposition of the AND gate provides little improvement in the delay through the gate. On the other hand, if the slacks of the outside inputs are large enough, the decomposition will lead to a significant improvement. A similar argument can be made for output signals from an ϵ-network which have fanouts in both the inside and the outside. If the slacks of the outside fanouts are large enough, a fanout optimization algorithm will be able to better isolate the critical paths .

In the second method, after weights are assigned to groups in the outside of the ϵ-network, a cut set is formed as done in the first method. One may ask what the difference is between the second method and the first method using a greater ϵ value. The answer is that we may face similar examples just described by simply increasing the ϵ value. On the other hand, we can control side inputs and outputs of an ϵ-network, by improving the delay of a cut set groups which is the outside of the ϵ-network, to improve the quality of decomposition at ϵ-network which is done after that.

In the actual implementation, these two methods are executed alternately in the main loop of our timing optimization procedure. The second method precedes the first one. Our experiments show that this procedure constructs faster circuits compared to the implementation using only the first method.

2.4 Correlation Coefficient

Before deciding which groups should be resynthesized, each group is assigned a weight which shows how feasible the group is if the group were resynthesized.

As described in later sections, our resynthesis algorithms will try to balance arrival times and required times within a group. In other words, they will try to make pins with early arrival times to have early required times while pins with late arrival times to have late required times. Therefore, a group which does not have balanced times would be improved by resynthesis and should be given a smaller weight.

To measure the balance between arrival times and required times, the following correlation coefficient is used.

$$P = \frac{\sum(A_i - M_A) * (R_i - M_R)}{\sqrt{\sum(A_i - M_A)^2 * \sum(R_j - M_R)^2}} \qquad (3)$$

- A is a arrival time
- R is a required time
- M_A is the average of arrival times
- M_R is the average of required times

From the above definition, if the times are already balanced within a group, then P=1, otherwise, $-1 <= P < 1$.

2.5 Resynthesis

The resynthesis algorithm for a random logic group is described in Figure 3. We illustrate this in Figure 4 with an example. In Figure 4.A, the input required times of the group, the present required time of the root Ro, and the fanout sets of the group with the required times are given.

First, the extracted group G1 (Figure 4.A) is transformed into technology independent form. Intermediate inverters are swept away to the inputs, and successive same-type gates, such as ANDs, ORs, or EXORs, are merged into a large same-type gate (Figure 4.B). Then these large gates are decomposed into 2 input gates(Figure 4.C) using a fanin ordering technique[1,10]. Figure 5 shows an example. For each node in the network, two signals which have earlier arrival times are selected and connected to a newly generated 2 input node. After that, the arrival time of the output of the 2 input node is calculated , and merged with the remaining arrival times. This is iterated until the network is fully decomposed into 2 input nodes. After the decomposition, provisional required times are set on all 2 input nodes by *put_requiredtime*() (Figure 4.D). The provisional required time at the 2 input node g is calculated as follows from the input side:

$$P_g = (R - MAX(Q_i, Q_j))/n + MAX(Q_i, Q_j) \qquad (4)$$

- Q_i, Q_j are arrival times at the inputs of g
- R is the required time of the root of the group
- n is the number of levels from the root

Finally, the tree mapping algorithm begins. All subtrees are processed from the input side. At each subtree, the best gate is selected from the library. A gate is considered to be the best along with already mapped subtrees if they can satisfy the provisional required time and the area is the smallest, or they cannot satisfy the required time but the arrival time is the smallest. Actually, the best gates are recorded for both polarities.

After the root is covered (Figure 4.F –actual result may differ.), previous tree covering algorithms[2-5] begin to trace back the tree and construct a mapped circuit. Before

```
resynthesis (G1)
        G2 = technology_independent (G1) ;
        G3 = merge_node (G2) ;
        G4 = decomposition_to_2 input (G3) ;
        G5 = put_required time (G4) ;
        G6 = tree_covering (G5) ;
        change_graph (G1, G6) ;
```

Figure 3 : Resynthesis

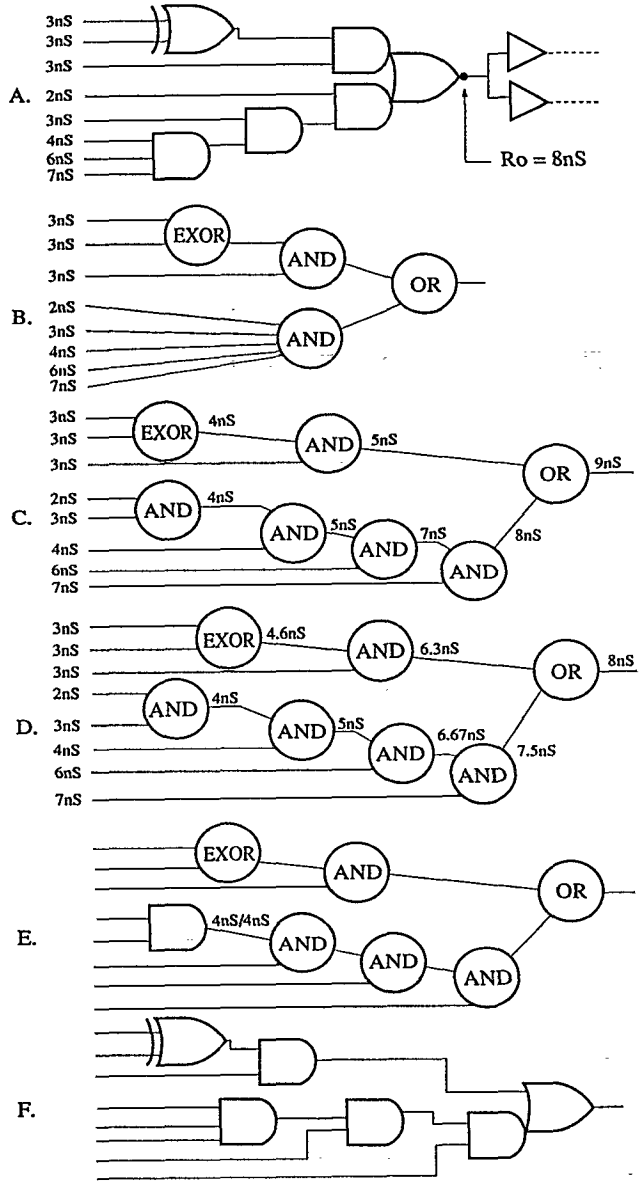

Figure 4 : An example of resynthesis

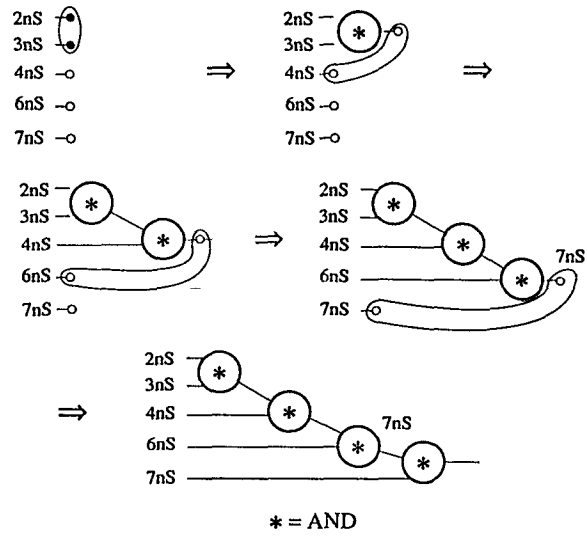

Figure 5 : Fanin ordering

doing so our algorithm selects the best gate along with a fanout tree. For each gate which can be used at the root, a fanout tree is generated and the slack at the root is checked. If the slack is positive, the gate and the fanout tree with minimum area are selected, otherwise the one with the maximum slack is selected.

For other types of groups, if there are several blocks, such as high power gate, which have the same functions as the root block, they are checked as a root with fanout trees. Otherwise, only fanout optimization is carried out.

2.6 Fanout Optimization

Our fanout optimization technique is derived from [8]. An informal description of the combinational merging algorithm is as follows. It begins with sorting a set of sinks in order of increasing required times. Then, it selects a buffer which drives a set of sinks with the latest required times. These sinks are deleted while the input pin of the buffer is added and put at the proper position with respect to its calculated required time. It repeats until the set becomes small enough to be driven by the source signal. As a result, sinks with early required times will be driven by the source signal or a buffer near the source, while sinks with late required times will not. In this sense, the combinational merging is able to isolate critical paths. An example is illustrated in Figure 6.

We can also handle dual signals by dividing sinks into two sets, P and N, which contain positive sinks and negative sinks, respectively. Our goal is to make both P and N small enough so that they can be driven by dual source signals. The sets P and N can be reduced by one of the following four operations:

1. Select a buffer which drives some sinks in P. Remove those sinks from P. Add the input of the buffer to P.
2. Select a buffer which drives some sinks in N. Remove those sinks from N. Add the input of the buffer to N.
3. Select an inverter which drives some sinks in P. Remove those sinks from P. Add the input of the inverter to N.
4. Select an inverter which drives some sinks in N. Remove those sinks from N. Add the input of the inverter to P.

These operations are treated as the branching operations of a branch and bound algorithm which works as follows.

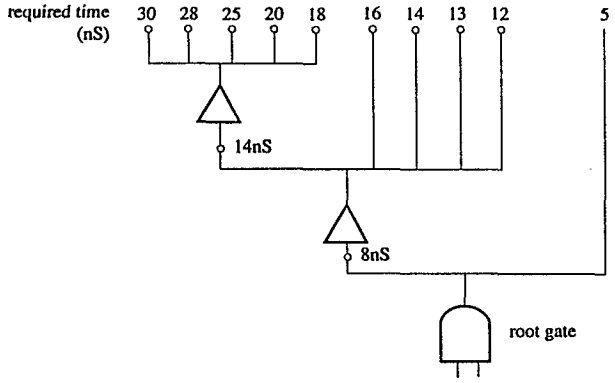

Figure 6 : Fanout tree

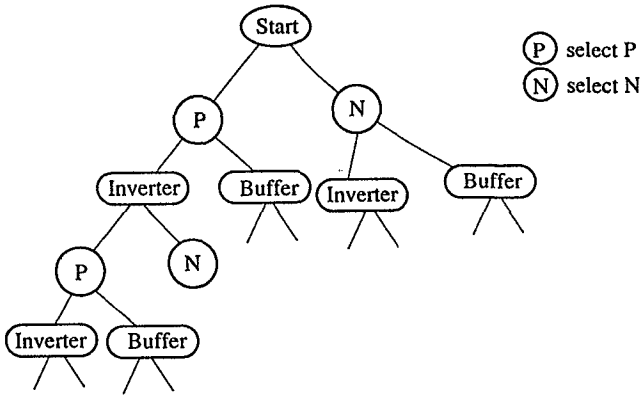

Figure 7 : A search space of the fanout optimization

Initial fanout trees are generated by applying one of the four branchings repeatedly. Then backtracking occurs. At each revisited intermediate state of P and N, another branch is selected to generate another fanout tree. If better trees are generated, they are recorded as the best trees so far. Some branches can be cut (bounded) if the intermediate result is worse than the best trees. Figure 7 shows a solution space explored by this algorithm.

Before each branching, a buffer or an inverter must be selected from the library. The number of sinks driven by the buffer must be decided, as well. If the number of buffers and inverters as well as the size of P and N are trivially small, these decisions might be treated as branching operations. But we must face some large fanouts. Therefore the above decisions are made in a greedy fashion. At each state, all combinations of a buffer or an inverter, P or N, and the number of sinks are tried and the required times at the input of the buffer or inverter are estimated. A combination is viewed as the best choice if it results in the largest reduction of P or N provided the required time of the input is larger than the smallest required time in the previous state. Otherwise, the combination which has the largest required time is regarded as the best.

2.7 Minimum Delay Assurance

Three types of transformation(Figure 8) are used to satisfy minimum delay constraints. The first one, which causes no area increase, is a reconnection of a net to a pin which has a later arrival time. This method is used if the slack of

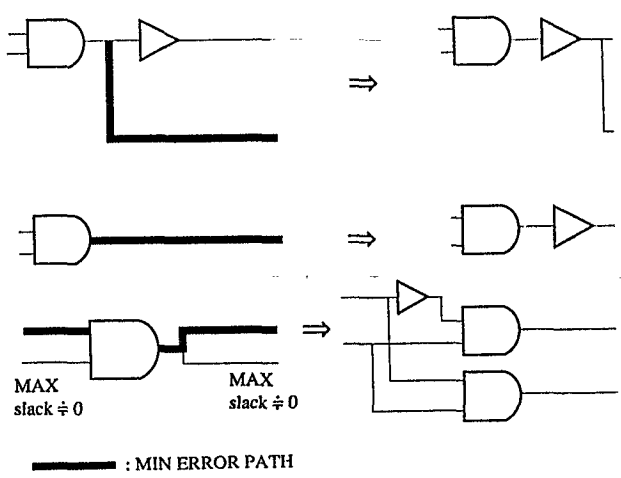

Figure 8 : Minimum delay assurance

the maximum delay is still positive after the transformation. The second one is delay gate insertion which can be applied provided slack of the maximum delay also remains positive. The third one is a combination of gate duplication and delay gate insertion. This method is used when it is impossible to satisfy both constraints (max delay and minimum delay)at the same time by the first two methods.

The above transformations are used in two different ways, called group searching, and path tracing. Group searching checks each group's root and fanout tree to see if a transformation can be carried out without maximum delay violation. The transformation is carried out if possible. If some minimum delay errors still exist after group searching, path tracing is tried. As mentioned earlier, required times of the input pins of the flip-flops are calculated using the maximum clock skew. But path tracing uses the specific skew between two flip-flops. It traces each path from an input pin of a flip-flop to the output pins of source flip-flops. When it reaches an output of a flip-flop, it checks if the minimum delay through the path is large enough to compensate for the skew and the hold time of the load flip-flop. If so, it does nothing but checks other paths. Otherwise, it adds some delay on feasible places on the path which have relatively large slacks for maximum delay constraints.

3 Results

The above techniques have been implemented in a program running on NEC EWS4800 workstations. We experimented on netlists of the DAC'86 benchmark set[13] and circuits taken from production chips for our main frame computers. After being synthesized by FUSION[11] using the NEC CMOS5 gate array library, the resulting netlists were fed to the timing optimization program.

The timing constraints were set to 0 ns for primary outputs and 0 ns as the clock cycle ; hence the program tries to speed up the circuits as much as it can. ϵ was set to 3 ns.

Table 1 shows the results of maximum delay improvements. The first half of Table 1 shows the results for the benchmark circuits and the bottom is on production circuits. On the average, delay was reduced by 37% while the area increase was 27%. Without the complementary method in the bottleneck analysis, delay reduction was only 28% and area increase was 28% on the average. Only rd53 and vg2 were improved better without the complementary method.

CIRCUIT	using the complementary method		without the complementary method	
	AREA ↑	DELAY ↓	AREA ↑	DELAY ↓
bw	38	40	38	27
duke2	34	41	33	9
f2	33	30	33	30
rd53	28	31	39	38
rd73	34	32	34	25
sao1	33	50	23	11
sao2	28	36	34	34
vg2	18	27	16	38
5xp1	29	44	27	21
9sym	29	52	30	44
Actual				
data1	6	22	6	19
data2	34	38	35	30
data3	15	37	15	32
data4	22	40	28	31
average	27	37	28	28

Table 1: Results of timing optimization

AREA ↑ increase in area
DELAY ↓ decrease in circuit delay

4 Conclusions and Future Work

We have presented new timing optimization techniques including bottleneck analysis, tree covering, fanout optimization, and minimum delay assurance. These have been implemented and integrated within the FUSION logic synthesis system. The results on the benchmark set and production circuits show significant improvements of their performance.

We are now improving some of the algorithms to work more efficiently. For example, we are incorporating balanced tree buffering techniques which are parts of [6,9] as a bounding operation in our fanout optimization algorithm.

Other plans include more dynamic restructuring techniques which can change the topology of a gate level DAG as well as a group level DAG, a more sophisticated approach on the complementary method of the bottleneck analysis, and adaptation to other clocking schemes.

5 Acknowledgments

We would like to thank Kazutoshi Takahashi for giving us the opportunity to develop the system and Yoshikazu Kamogawa, Toyotaka Nishiki, and Hisako Kato for their implementation efforts. We are also thankful to Naotaka Maeda for supplying us with the benchmarks in readable formats to our system.

We are grateful to Professor Robert Brayton, Dr. Hervié Touati, Mr. Kanwar Jit Singh, and Professor Patrick McGeer for their valuable comments on the first version of this paper and their stimulating discussion on the future work.

References

[1] K.J. Singh, A.R. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Timing Optimization of Combinational Logic", ICCAD-88, pp.282-285, 1988.

[2] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli and A.Wang, "Technology Mapping in MIS", ICCAD-87, pp.116-119, 1987.

[3] K.Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching", 24th Design Automation Conference, pp.341-347, 1987.

[4] K.Keutzer and M.Vancura, "Timing optimization in a Logic Synthesis system", Proceedings of International Workshop on Logic and Arch. Synthesis for Silicon Compilers, pp.1-13, May 1988

[5] H.J.Touati,C.W.Moon,R.K.Brayton, and A.Wang, "Performance Oriented Technology Mapping", Proceedings of the sixth MIT VLSI Conference, pp.79-97, 1990.

[6] K.J.Singh and A.Sangiovanni-Vincentelli, "A Heuristic Algorithm for the Fanout Problem", 27th Design Automation Conferance, pp.357-360, 1990.

[7] C.L.Berman,J.L.Carter,and K.F.Day, "The Fanout Problem:From Theory to Practice", Advanced Research in VLSI:Proceedings of the 1989 Decennial Caltech Conference, pp.69-99, 1989.

[8] M.C. Golumbic, "Combinational Merging", IEEE Transactions on Computers,25(11), pp.1164-1167, 1976.

[9] H.J.Touati, "Performance Oriented Technology Mapping", UCB/ERL M90/109, UC Berkeley, Nov. 1990

[10] J.A. Darringer, D. Brand, W.H. Joyner, and L. Trevillyan, "LSS: A System for Production Logic Synthesis", IBM J.Res.Develop.28(5), pp.537-545, 1984

[11] T.Yoshimura and S.Goto, "A RuleBased Algorithmic Approach for Logic Synthesis", ICCAD-86, pp.162-165, 1986.

[12] R.B. Hitchcocke,Sr., G.L. Smith and D.D. Cheng, "Timing Analysis of Computer Hardware", IBM J.Res.Develop. 26(1), pp.100- 105, 1982.

[13] A.J.de Geus, "Logic Synthesis and Optimization Benchmarks for the 1986 Design Automation Conference", 23rd Design Automation Conferance, pp.78, 1986.

[14] D.E.Wallace, M.S.Chandrasekhar, "High-Level Delay Estimation for Technology-Independent Logic Equations", ICCAD-90, pp.188-191, 1990.