Simultaneous Gate Sizing and Fanout Optimization

Wei Chen[*], Cheng-Ta Hsieh[+], Massoud Pedram[*]
[*]University of Southern California, Los Angeles, CA 90089
[+]Verplex Systems, Inc., Milpitas, CA 95035

Abstract

This paper describes an algorithm for simultaneous gate sizing and fanout optimization along the timing-critical paths in a circuit. First, a continuous-variable delay model that captures both sizing and buffering effects is presented. Next, the optimization problem is formulated as a non-convex mathematical program. To manage the problem size, only a small number of critical paths are considered simultaneously. The mathematical program is solved by a non-linear programming package. Finally, a design flow based on iterative selection and optimization of the k most critical paths in the circuit is proposed. Experimental results show that the proposed flow reduces the circuit delay by an average of 9.2% compared to conventional flows that separate gate sizing from fanout optimization.

1 Introduction

Timing constraints in modern VLSI circuits are becoming increasingly tighter. Gate sizing and fanout optimization techniques are widely used to meet these constraints. Gate sizing reduces the circuit delay by adjusting the gate sizes and hence their drive strengths and input capacitances. Fanout optimization achieves circuit delay reduction by speeding up the timing-critical signals through insertion of sized buffers.

Gate sizing methods can be classified into two categories: discrete and continuous. Discrete sizing methods only allow a set of discrete sizes for each gate. They use combinatorial algorithms or stochastic search to determine the best size for each gate [1]. Continuous sizing methods on the other hand assume that gate sizes are continuous variables and then use mathematical programming to formulate and solve the optimization problem [2]. Continuous sizing methods have a more global view of the solution space and hence tend to achieve better initial results. The final quality may however degrade after the round off step, which is required because in reality there are only discrete sizes allowed for each gate in the ASIC library. In today's ASIC design process however, the number of available gate sizes in standard gate libraries is increasing, so the rounding error is becoming smaller. Furthermore, the advent of on-the-fly-synthesized gate libraries is helping to alleviate this problem further.

Fanout optimization methods are usually applied in a circuit one net at a time. They can again be divided into discrete buffer sizing and continuous buffer sizing. The discrete buffer size-based fanout optimization problem has been proven to be NP-complete [3]. Most of the previous work in this category hence assumes a fixed template for the buffer tree [4]. Using a buffer library with continuous sizes greatly simplifies the fanout optimization problem [5].

The Semiconductor Research Corporation under contract no. 98-DJ-606 sponsored this research.

Traditionally, gate sizing and fanout optimization are done individually and at different stages in the design process. This sequential flow can adversely affect the circuit performance as illustrated in the example below.

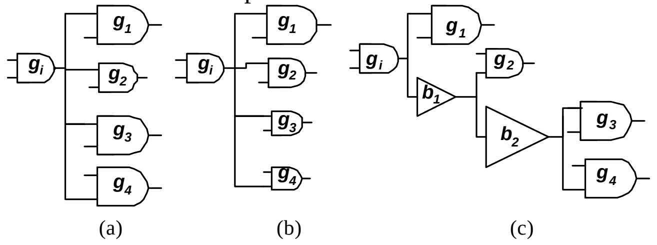

Figure 1. Motivation for simultaneous gate sizing and buffer insertion.

In Figure 1 (a), assume that g_i, g_1 lie on a timing-critical path. (Object sizes in the schematic represent the actual gate sizes in the circuit.) Furthermore, assume that the required times for g_1 to g_4 are in increasing order. Starting from this configuration, a gate sizing tool will likely size down the non-critical sinks, g_2 to g_4 to improve the critical path's timing as shown in Figure 1(b). On the other hand, starting from the configuration in Figure 1 (a), a fanout optimization tool will likely build the buffer tree shown in Figure 1 (c) to isolate the non-critical gates from g_i.

It is possible that in Figure 1 (b) even though g_3, g_4 are sized down to their minimum allowed sizes in the library, their output arrival times are still earlier than their required times. So in fact, as in Figure 1 (c), buffer b_1 can be inserted to improve the arrival time at output of g_1. Similarly, in Figure 1 (c), if g_3, g_4 are sized down, buffer b_2 can be removed without violating the timing requirement at output of g_2. The gate-sizing tool cannot however add b_1 in the same way that the fanout optimization tool cannot size down g_3, g_4. From this example, we can see the shortcomings of separating the gate sizing and fanout optimization steps. Because each step tries to make use of all the freedom in the optimization space, it does not leave much optimization opportunity for the other. At the same time, each step is limited in the kind of optimization that it can perform. By combining these two steps into one integrated step, we enlarge the solution space and achieve more optimized results.

An interleaved buffer insertion and transistor-sizing algorithm is proposed in [6]. The algorithm evaluates the effect of buffer insertion and gate sizing separately and implements the one that improves circuit delay best. Experimental results demonstrate that even this greedy approach outperforms those that only do gate sizing.

This paper presents an algorithm for continuous-variable simultaneous gate and buffer sizing. The resulting problem formulation become a non-convex mathematical program, hence the size of the mathematical program must be controlled carefully to avoid excessive runtimes. This is achieved by restricting the number of timing-critical paths that are considered at one time. The whole circuit is in turn optimized iteratively by a sequence of timing recalculation and simultaneous sizing and buffering.

The rest of this paper is as follows. Delay model is described in Section 2. Precise problem formulation is given in Section 3. Detailed algorithm and flow are presented in Section 4. Experimental results and conclusions are given in Sections 5 and 6, respectively.

2 Delay Model

2.1 Notation

The following notation is used in this paper:

g_i	gate i
a_i	arrival time of g_i's output
r_i	required time of g_i's output
z_i	size of g_i
$buf_{i,j}$	buffer chain inserted from g_i to its sink g_j
$h_{i,j}$	gain of the buffer chain $buf_{i,j}$
$x_{i,j}$	number of levels of $buf_{i,j}$
$cbuf_{i,j}$	input capacitance of the first gate in $buf_{i,j}$
$dint_{i,j}$	intrinsic delay of g_j for a signal transition from the output of g_i
$rdr_{i,j}$	driving strength of g_j for a signal transition from the output of g_i
$cload_j$	sum of input capacitances of g_j's fanout gates
cin_i	input capacitance of g_i
$d_{i,j}$	delay from the output of g_i to the output of g_j
$dgate_{i,j}$	delay from the output of $buf_{i,j}$ to the output of g_j
$dbuf_{i,j}$	delay of the buffer chain $buf_{i,j}$
$C(k)$	set of gates on the k most-critical paths
$Ne(k,i)$	set of gates that are direct fanout gates of $C(k)$

2.2 Timing Analysis

Let directed graph $G(V, A)$ represent the net list of a circuit. The vertex set V represents the set of gates in the circuit whereas the edge set A represents the source-to-sink connections among gates. Associated with each gate g_i in the circuit, there exist an actual output arrival time a_i and a required output arrival time r_i. The circuit designer specifies arrival times for circuit inputs and required times for circuit outputs from chip level considerations.

The arrival time a_j is given by:

$$a_j = max\{(a_i + d_{i,j}) \mid \forall(v_i, v_j) \in A\}$$

The required time r_i is given by:

$$r_i = min\{(r_j - d_{i,j}) \mid \forall(v_i, v_j) \in A\}$$

where $d_{i,j}$ is the delay from the output of g_i to the output of g_j. g_i's slack time s_i is defined $s_i = r_i - a_i$. A (timing) *critical path* is a path in which the sequence of vertices $(v_i,\ldots,v_o)$ that comprise the path ($v_i \in$ primary inputs and $v_o \in$ primary outputs), all have slack values less than or equal to zero.

2.3 Gate Sizing Delay Model

The continuous-variable pin-dependent gate delay model of [7] is adopted. [1]

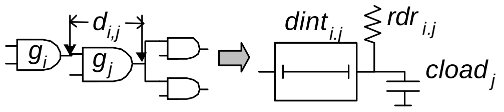

Figure 2. Gate delay model.

$$d_{i,j} = dint_{i,j} + rdr_{i,j} \cdot cload_j \tag{1}$$

$dint_{i,j}$ represents the intrinsic delay of g_j for a signal transition from the output pin of g_i. For gates with the same logic function, $dint_{i,j}$ is nearly a fixed value, independent of the gate size. $rdr_{i,j}$ stands for the drive strength of g_j for a signal transition from the output pin of g_i. $cload_j$ is the input capacitance of the gate load of g_j. $cload_j = \sum_{g_k = fanout(g_j)} cin_k$. $rdr_{i,j}$ and cin_k are functions of the gate size and gate function. Using linear regression, we empirically obtain the following equations:

$$rdr_{i,j} = a_1 + b_1 / z_j$$
$$cin_j = a_2 + b_2 \cdot z_j$$

Equation (1) can then be rewritten as:

$$d_{i,j} = dint_{i,j} + rdr_{i,j}(z_j) \cdot \sum_{g_k = fanout(g_j)} cin_k(z_k)$$

2.4 Buffer Insertion Delay Model

The global fanout optimization problem in conventional logic synthesis flow is solved net by net by applying a local fanout optimization algorithm. The latter problem can be expressed as:

Given a source g_i with arrival time a_i and a set of sink g_j with capacitance load cin_j, polarity P_j and required time r_j, find the optimum topology of buffer tree and the appropriate size for each inserted buffer to minimize the load "seen" by the source g_i, such that the arrival time of g_j is less than r_j.

2.4.1 Buffer Chain Model

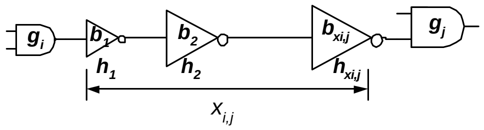

Figure 3. Single sink buffer chain.

As shown in Figure 3, buffers $buf_{i,j}$ inserted on the link from the output of g_i to its single sink g_j, consist of $b_1, b_2,\ldots,b_{xi,j}$, where $x_{i,j}$ denotes the number of inserted buffers between g_i and g_j. To calculate the delay of $buf_{i,j}$, denoted as $dbuf_{i,j}$, the logical effort based delay model [8] is used. This model is a reformulation of the conventional RC model of CMOS gate delay. The delay of buffer b, $d=t(p+gh)$.[2] p is the parasitic delay of the gate. g is called the *logical effort* of the gate and depends only on the topology of the gate and its ability to produce output current. h is

[1] For simplicity, the interconnect delay has been ignored in this formula. It is however easy to extend this formula to use a statistical wire load model based on the pin-count of the net and size of the circuit.

[2] t is a scaling parameter that characterizes the semiconductor process being used. It converts the unit-less quantity $(p+gh)$ to d, which has time units. For simplicity and without loss of generality, we will drop t in the discussion that follows.

called *electrical effort* (or gain), which is defined as $load/c_{in}$. p and g are independent of the buffer sizes while c_{in} is the input pin capacitance of the buffer. In Figure 3, $h_1, h_2, \ldots h_{xi,j}$ are gains of the buffer $b_1, b_2, \ldots, b_{xi,j}$, respectively. Suppose the input capacitance of g_j is cin_j. The input capacitance of the first buffer b_1 is $c_1 = cin_j / \prod_{l}^{x_{i,j}} h_l$.

***Theorem* [9]** Under the required time constraint $a_i + \sum_i (p + g \cdot h_i) < r_j$ for the sink, c_1 is minimized when $h_1 = h_2 = \ldots = h_{xi,j} = h_{i,j}$.

In this paper, we take advantage of this theorem, since by minimizing the load of g_i, the arrival time of g_i is shortened, and its driver gates are sped up. Notice that the delay from the output of g_i to the output of the last level buffer $b_{xi,j}$ is:

$$dbuf_{i,j} = x_{i,j}(p + g \cdot h_{i,j}) \qquad (2)$$

2.4.2 Buffer Tree Model

Without information about the topology of the buffer tree, the delay from the net source to each net sink cannot be calculated correctly. Assume that all buffer sizes are available in the cell library, the buffer tree can be manipulated by the merge and split operations without affecting the optimality of the buffer tree [9]. These operations are illustrated in Figure 4.

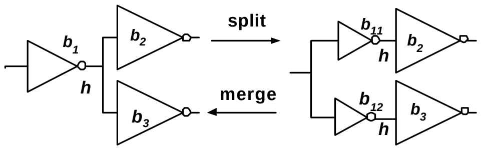

Figure 4. Buffer tree merge and split transformations.

***Theorem* [9]** If gains of b_1, b_{11}, b_{12} are the same, then the timing and input capacitance properties are preserved by the merge/split transformations (cf. Figure 4).

As a result, the optimal fanout tree with appropriate buffer sizes may be split into a fanout-free tree, which is composed of a set of buffer chains connected at the source of the net. The reverse is obviously true too. Hence, we can build the buffer chains separately and then merge them to obtain the optimal fanout tree.

Equation (2) can be extended to multiple sink buffer trees as shown in Figure 5. Recall that for each sink g_j of g_i, $h_{i,j}$ is the gain of every buffer in $buf_{i,j}$.

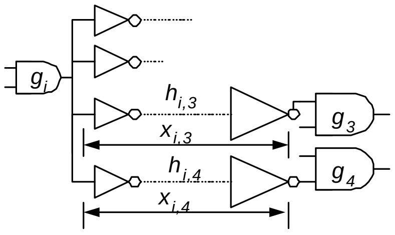

Figure 5. Multiple sink buffer tree.

2.5 Simultaneous Gate Sizing and Buffer Insertion Delay Model

To express simultaneous gate sizing and fanout optimization problem in a mathematical form, the delay model must reflect the effect of size change and possible insertion of a buffer chain.

As shown in Figure 6, we combine the gate sizing and the buffer chain delay models. Delay $d_{i,j}$ is divided into two parts: $dbuf_{i,j}$, which is the delay from g_i's output to the buffer chain's output and $dgate_{i,j}$, which is the delay from the buffer chain's output to g_j's output.

As before, $dbuf_{i,j}$ is calculated by Equation (2). Notice however that previously output load of the buffer chain cin_j was a known value whereas now cin_j changes with g_j's size. The load of g_j is not determined from its direct fanout gates, instead it is determined from the input capacitance of the very first buffer in the buffer chain: $cbuf_{j,k}$: $cbuf_{j,k} = \frac{cin_k}{(h_{j,k})^{x_{j,k}}}$

The complete set of delay equations is thus summarized as:

$$\begin{aligned} dbuf_{i,j} &= x_{i,j} \cdot (p + g \cdot h_{i,j}) \\ dgate_{i,j} &= dint_{i,j} + rdr_{i,j}(z_j) \sum_k \frac{cin_k(z_k)}{(h_{j,k})^{x_{j,k}}} \\ d_{i,j} &= dbuf_{i,j} + dgate_{i,j} \end{aligned} \qquad (3)$$

where k denotes the index of the fanout branch of g_j.

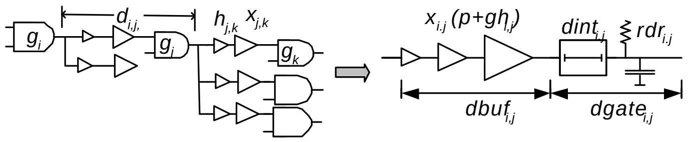

Figure 6. Gate delay model with buffer chains.

Theorem The delay model $d_{i,j}$ of Equation (3) is non-convex.

Proof Hessian matrix F of function f is the matrix of the 2^{nd} partial derivatives of f. Function f is convex over a convex set W containing an interior point if and only if the Hessian matrix F of f is positive semi-definite throughout W [11]. Readers can easily verify that $d_{i,j}$ given in equation (3) is not positive semi-definite. Therefore, the delay mode is non-convex. ■

Equation (3) describes the timing relations in the mathematical formulation of the simultaneous gate sizing and fanout optimization problem (section 3). Note when $x_{i,j}$ is equal to 0, it means no buffer is inserted between g_i and g_j, and $d_{i,j}$ becomes exactly the same as Equation (1). This model consistency is of course important, because we do not assume any buffer tree template before the solution is attempted, and we do not know whether or not an inserted buffer chain $buf_{i,j}$ exists. We let the mathematical programming package determine the value of $x_{i,j}$, and $h_{i,j}$, that is the best topology and size of the buffers trees. If Equations (1) and (3) were not consistent at $x_{i,j}$=0, for the edges with a zero-value buffer level, the real delay calculated by equation (1) would be different from the timing estimation of constraints formulated based on (3). The convergence of problem solution would therefore not be guaranteed. Other important properties of Equation (3) are its continuity and differentiability, which are indispensable to most mathematical programming packages.

3 Problem Formulation

3.1 Global Formulation

We would like to capture the timing relations in the whole circuit in one formulation, because such formulation would result in a globally optimized solution. The problem is stated as:

$$
\begin{array}{lll}
\text{minimize} & cycle & \\
\text{s.t.} & a_i \geq T_{start} & \forall v_i \in PI \\
& a_i \leq cycle & \forall v_i \in PO \\
& a_j \geq a_i + dbuf_{i,j} + dgate_{i,j} & \forall (v_i, v_j) \in A
\end{array} \quad (4)
$$

where T_{start} is the latest arrival time of all the primary inputs. In this formulation, for each gate there are two variables corresponding to its arrival time and gate size; for each edge, there are two variables corresponding to the number of inserted buffers and the buffer gain (recall that all buffers in the same buffer chain have identical electrical effort, i.e. identical gain). Suppose the number of gates is n and the number of edges is e. There are $(2n+2e)$ variables. The number of constraints is also e.

Observation: Equation (4) is a non-convex problem because $dgate_{i,j}$ is a non-convex function.

Each constraint of Equation (4) is related to quite a small number of variables: a_i, a_j, $x_{i,j}$, z_j, z_k, $x_{j,k}$ and $h_{j,k}$. So the problem formulation is very sparse. *LANCELOT* [12] is especially effective in solving this kind of large-scale, non-linear, sparse problem. It has been adopted in many VLSI CAD tools and shows robustness and high efficiency.

We use *LANCELOT* to solve Equation (4) directly on several benchmark circuits. Although *LANCELOT* shows good performance on this kind of problem, Equation (3) has, in worst case, $O(n^2)$ variables and constraints. Furthermore, the delay model is non-convex. These considerations make the global optimization formulation infeasible in practice for large circuits.

3.2 Critical Section Formulation

Instead of optimizing the whole circuit in one shot, we can iteratively optimize the k most-critical paths of the circuit [7]. *C(k)* is defined as the set of gates on the k most-critical paths in the circuit. *Ne(k)* is defined as the set of gates which are the immediate fanouts of *C(k)*. In each iteration, *C(k)* and *Ne(k)* are identified. We only focus on optimizing them. The operations performed include gate sizing *C(k)*, fanout optimization of *C(k)* and gate sizing of *Ne(k)*. Compared to only gate sizing, the topology of the critical paths is not fixed. Compared to local fanout optimization, the sinks of the critical paths are sizable, and the buffer trees are generated on the basis of whole path delay, not for a single net.

By carefully controlling the boundary conditions, that is the arrival times of *Ne(k)*, solution convergence is guaranteed [7]. Note that only the gates in *C(k)* and *Ne(k)* are changed, all others are fixed, therefore the load of gates in *Ne(k)* are not changing. So if we guarantee that the arrival time of *Ne(k)* after optimization is no larger than the specified required time, the arrival time of the gates outside of *C(k)* and *Ne(k)* will not increase. This analysis ignores the reconvergent fanout issues, and hence holds only approximately. In practice however, enforcing boundary constraints for *Ne(k)* is quite effective .

The new formulation is given as:

$$
\begin{array}{lll}
\text{minimize} & cycle & \\
s.t. & a_i \geq T_{start} & \forall v_i \in PI, v_i \in C(k) \\
& a_i \leq cycle & \forall v_i \in PO, v_i \in C(k) \\
& a_j \geq a_i + dbuf_{i,j} + dgate_{i,j} & \forall (v_i, v_j) \in A, v_i \in C(k) \\
& a_j \leq d \cdot r_j & \forall v_j \in Ne(k)
\end{array} \quad (5)
$$

where $dbuf_{i,j}$ and $dgate_{i,j}$ were given in Equation (3). d is a parameter to control the strictness of the arrival time requirement on *Ne(k)*. Its value is set to less than or equal to 1.

We define *critical edge* as an edge in graph *G(V, A)* that is driven by a gate in *C(k)*. Suppose there are *n'* gates in *C(k)*, which introduce *e'* critical edges and *m'* gates in *Ne(k)*. There are *2(n'+e'+m')* variables (arrival time and size for each gate in *C(k)* and *Ne(k)*, buffer chain level and gain for each critical edge) in Equation (5)). In this way the problem size decreases. If the circuit is small, we can increase k to put more gates in *C(k)*.

4 Algorithm

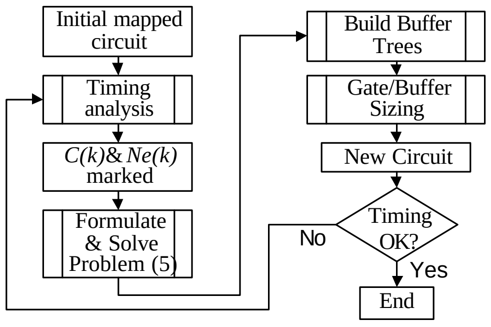

Figure 7 Algorithm flow.

The algorithm flow is depicted in Figure 7. First, timing analysis is performed on the circuit network. The k most-critical paths are marked. The buffer trees, which are driven by *C(k)* and built in previous iteration, are removed such that the new buffer trees can be constructed from Equation (5). The rationale for removing previously constructed buffer trees is that in this way we allow deleting redundant or non-optimal buffer trees. Next, problem formulation (5) is generated and passed on to the *LANCELOT* package. *LANCELOT* produces gate sizes, buffer chain lengths and individual buffer gains. The buffer tree for each gate in *C(k)* is formed by recursive merging of the buffer chains on that net (c.f. section 4.2). After the fanout tree topology is decided, the algorithm determines the buffer and gate sizes. In the end, a new circuit net list is generated. The above steps are repeated until the timing constraints are met.

4.1 Buffer Tree Generation

After Equation (5) is solved, $x_{i,j}$ is usually a non-integral value. In reality, a feasible solution should be an integer. Suppose m_1 and m_2 are the two nearest feasible integers considering polarity requirement of $x_{i,j}$. We round $x_{i,j}$ to the number that satisfies the required timing constraint of g_j. If both values meet (or violate) the required time demand, we pick up the value that makes $cbuf_{i,j}$, the input capacitance of the first gate in the tapered buffers $buf_{i,j}$, smaller. This heuristic keeps the load of the critical gate smaller, thus reducing the arrival time of critical gate.

After the number of levels for each buffer chain is determined, the size of buffers are calculated from its level and gain. The size of these buffers is again in general a non-integral value, and indeed some sizes may be less than one. The merge operation is done recursively from the first level to reduce the number of buffers and increase their sizes to make them whole buffers. The advantage of merge is that it can minimize the round up error due to non-integral buffer sizes and at the same time reduce the buffer areas. Since the gain of each chain is calculated for different sinks separately, their values may not be same. The merge transform keeps the delay unchanged only when two branches have the same gain. Therefore, we define a constant ε and merge two buffers as long as the difference of their gains is less than or equal to e.

5 Experimental Results

Our algorithm was implemented and run on Pentium-III 733MHz machine. Table 2 shows our experimental results for performing global optimization on some benchmark circuits. These results correspond to the solution to Equation (4). The initial cell count and delays for all circuits are given in Table 1. Initially, each logic gate is mapped to the corresponding minimum size cell in the library. To make the comparison fair, we iteratively perform both buffer + sizing (B+S) and sizing + buffer (S+B). Notice that buffers inserted in iteration *i* are kept during sizing in iteration *i+1*, but they are removed before buffering in iteration *i+1*. The delays of the first four iterations are compared with the delay of (one-step) simultaneous buffer/sizing (B/S). The gate sizing and fanout optimization techniques are described in [7] and [10], respectively. The B+S and S+B iterations converge to the final circuit delay only after two iterations. The percentage improvement of B/S over B+S or S+B is calculated as the delay of B/S divided by the smaller of the two delays obtained by B+S and S+B. The improvement of B/S over the sequential methods is an average of 5.1%.

The global formulation is too expensive (and indeed impractical) to apply to large circuits. Table 3 presents results of the iterative optimization method based on Equation (5). In each iteration, we choose a *k* value such that the *k* most critical paths consist of about 150 gates. For the sequential methods, we perform two local iterations of B+S or S+B on the gates in the critical section whereas for the B/S technique, we solve Equation (5) in one shot. Circuit timing is updating from one iteration to next. Examining the results for the first four circuits in Tables 2 and 3, we note that the delay of the critical section formulation B/S is only a little bit larger than that of the global formulation B/S. Sequential methods however perform worse using the critical section formulation flow. Percentage improvement of the B/S technique over sequential techniques becomes more pronounced. The average delay improvement is 9.2%.

Circuit	Cell	Delay (Initial)	Circuit	Cell	Delay (Initial)
C499	232	9.92	C3540	573	24.86
C1908	262	13.13	k2	589	17.62
C880	282	13.24	C5315	849	16.45
C1355	375	12.83	C7552	1391	31.18
dalu	513	21.97			

Table 1. Benchmarks information.

Circuit		Iterations				Delay (B/S)	CPU (sec)
		1	2	3	4		
C499	D (B+S)	5.35	5.02	5.06	5.03	4.82	86
	D (S+B)	5.13	5.08	5.03	5.04		
C1908	D (B+S)	7.51	7.37	7.40	7.40	7.08	144
	D (S+B)	7.36	7.27	7.35	7.36		
C880	D (B+S)	7.67	7.51	7.54	7.50	7.22	201
	D (S+B)	7.59	7.55	7.55	7.54		
C3540	D (B+S)	10.54	10.39	10.35	10.37	9.45	1821
	D (S+B)	10.58	10.41	10.32	10.38		

Table 2. Global formulation results.

Circuit	Delay (B+S)	CPU (B+S)	Delay (S+B)	CPU (S+B)	Delay (B/S)	CPU (B/S)	Improve (%)
C499	5.39	245	5.38	228	5.03	198	7.0
C1908	7.50	302	7.43	335	7.18	236	3.5
C880	7.94	438	7.90	394	7.34	255	7.6
C1355	7.38	391	7.44	457	6.95	432	6.2
dalu	13.46	459	13.40	865	12.82	693	4.5
C3540	11.03	827	11.12	734	9.81	250	12.4
k2	10.73	522	10.59	603	9.23	594	14.7
C5315	11.14	1894	11.23	2433	10.02	2097	11.2
C7552	18.15	4758	17.91	4458	15.54	3295	15.3

Table 3. Critical section formulation results.

6 Conclusions

In this paper, we introduced a new delay model for describing gate sizing with inserted buffers. The simultaneous gate sizing and fanout optimization problem was formulated as a non-linear programming problem and solved by *LANCELOT*. To control the problem size, we used an iterative flow to optimize the *k* most-critical paths. Merge and split operations were adopted to transform the fanout free tree to a general buffer tree. Experimental results showed that our simultaneous gate sizing and fanout optimization algorithm has an average delay improvement of 9.2% compared to conventional methods based on sequential fanout optimization and gate sizing flow.

Reference

[1] O. Coudert, R. Haddad, "New Algorithms for Gate Sizing: a Comparative Study", *Proc. of 33rd DAC*, pp.734-739, Jun 1996.

[2] M. Berkelaar, J. Jess, "Gate Sizing in MOS Digital Circuits with Linear Programming", *Proc. of European DAC*, pp.217-221, 1990.

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

[4] H. Touati, "Performance-oriented Technology Mapping", Ph.D. thesis, *University of California, Berkeley, Technical Report UCB.ERL M90/109*, November 1990.

[5] K. Kodandapani, J. Grodstein, A. Dominic, H. Touati, "A Simple Algorithm for Fanout Optimization using High-Performance Buffer Libraries", *Proc. of ICCAD*, pp. 466-471, November 1993.

[6] Y. Jiang, S. Sapatnekar, C. Bamji, J. Kim, "Interleaving Buffer Insertion and Transistor Sizing into a Single Optimization", *IEEE Transactions on VLSI Systems*, vol.6, No.4, pp. 625 - 633, December 1998.

[7] W. Chen, C. T. Hsieh, M. Pedram, "Simultaneous Gate Sizing and Placement", *IEEE Transactions on CAD*, Vol.19, No.2, pp.206-214, February 2000.

[8] I. Sutherland, R. Sproul, "The Theory of Logical Effort: Designing for Speed on the Back of an Envelope", *Advanced Research in VLSI*, Santa Cruz, 1991.

[9] D. Kung, "A Fast Fanout Optimization Algorithm for Near-Continuous Buffer Libraries", *Proc. of 35th DAC*, pp. 352-355, June 1998.

[10] P. Rezvani, A. Ajami, M. Pedram, H. Savoj, "Leopard: A Logical Effort-based fanout OPtimization for Area and Delay", *Proc. of ICCAD*, pp. 516-519, November 1999.

[11] D. Luenberger, *Linear and Nonlinear Programming*, *Addison-Wesley*, pp.180, 1984.

[12] A. R. Conn, N. I. M. Gould, P. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization*, Springer-Verlag, 1992.