Performance Driven Multiple-Source Bus Synthesis Using Buffer Insertion

Chia-Chun Tsai[+], De-Yu Kao[*], Chung-Kuan Cheng, and Ting-Ting Lin[*]

Department of Computer Science and Engineering
[*]Department of Electrical and Computer Engineering
University of California at San Diego, La Jolla, CA 92093

Abstract-- A heuristic algorithm for a given topology of a multiple-source and multiple-sink bus to reduce the signal delay time is proposed. The algorithm minimizes the delay by inserting buffers into the candidate locations and sizing the buffers. Experiments show up to 7.2%, 20.7%, and 29.6% improvement in delay for 2.0, 0.5, and 0.3 micron technologies, respectively.

I. Introduction

In the design of high-performance VLSI systems, buses inherently exist in the chip. A multiple-source bus is used to conserve routing area and number of terminals of functional blocks. However, the trade-off is a larger loading capacitance and delay time. Therefore, reducing the signal delay time in a multiple-source and multiple-sink bus is an important practical problem.

A conventional approach to reduce the delay between sources and sinks is sizing the source drivers. For a given number of sources, engineers manually tune each driver's size to reduce the delay times until the timing requirement is matched.

Many papers [1-6] concentrate on the analogous problem, transistor or gate sizing, in CMOS and digital circuits. They try to find the best sizes of a given number of gates/transistors and to reduce the propagation delay time with analytical or heuristic methods. Other researchers in [7-10] insert buffers into the wires to reduce the delay in single-source Steiner tree distribution.

The multiple-source bus is complicated by its multi-source and multi-sink characteristics. The best solution for a particular source and its sinks may result in another source not fulfilling its timing requirements. This paper is the first effort to reduce multiple-source bus signal delay by buffer insertion.

Given a set of N candidate locations for buffer insertion and timing requirements, our goal is to find a set of locations for buffers and their sizes to minimize delay. An exhaustive search of all the combinations of location and sizes of the inserted buffers is not feasible for industry applications.

We adopt the λ-optimal approach by Lin and Kernighan [11, 12] to search for the buffer assignment. We observe that sizing of buffers belongs to the class of geometric programming problems. A heuristic iterative improvement method is used to insert buffers into the candidate locations and to tune the buffer size. Our algorithm takes the running time of $O(N^4 n \ln W_{max})$, where N is the number of candidate buffer locations, W_{max} is the maximum buffer size and n is the total number of sources/sinks. The results show that averages of 7.2%, 20.7%, and 29.6% time delay improvement for 2.0, 0.5, and 0.3 micron technologies respectively.

The remainder of the paper is organized as follows. Section II describes the problem statement. Section III gives the wire and buffer delay models. The buffer insertion algorithm and its time complexity are introduced in Section IV. The last two sections present the experimental results and conclusions.

II. Problem Statement

We first define the following symbols used through this paper.

$\boldsymbol{p_i}$**:** The ith terminal in a bus; a terminal may be a source, sink, or both.
$\boldsymbol{n}$**:** number of terminals in the bus.
$\boldsymbol{t_{ij}}$**:** delay time from source i to sink j without buffer insertion.
$\boldsymbol{a_{ij}}$**:** actual arrival time from source i to sink j with buffer insertion.
$\boldsymbol{N}$**:** number of candidate locations for buffer insertion.
$\boldsymbol{W_{max}}$**:** maximum buffer size allowed.
$\boldsymbol{W_{min}}$**:** minimum buffer size allowed; we set $W_{min} = 0$ to denote no buffer inserted.
$\boldsymbol{w_{bi}}$**:** buffer size assigned to buffer (source or sink) i.
$\boldsymbol{R_w}$**:** wire resistance per unit length.
$\boldsymbol{R_b}$**:** output resistance of a unit size buffer.
$\boldsymbol{C_w}$**:** wire capacitance per unit length.
$\boldsymbol{C_b}$**:** The input capacitance of a unit size buffer.
$\boldsymbol{l(f,g)}$**:** length of wire (f,g).
$\boldsymbol{D_b}$**:** intrinsic delay of an inverter.

A bus consists of n terminals and N bus-wire segments. A terminal may be a source or sink in different timing periods. In each timing period, there can be only one working source but multiple sinks. In the same period, a subset of the wire segments is used.

Fig. 1 shows an example of a six-terminal bus with 14 wire segments and four timing periods. In the figure, terminal p_1 is the source in periods 1 and 3. Terminals p_2 and p_4 are the sources in periods 2 and 4, respectively. Note that three sources, p_1, p_2, and p_4 are also sinks in the different timing periods. Each source has at least one sink. For instance, source p_4 has four sinks, p_1, p_3, p_5, and p_6 in the timing period 4. There are 3 source drivers (p_1, p_2, and p_4) and 14 possible buffer locations with one location on each wire segment.

Given the required time t_{ij} and the actual arrival time a_{ij}

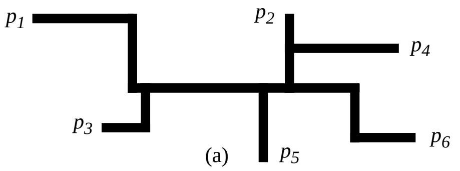
[+]C.C. Tsai is on leave (7/94-7/95) from the Dept. of Electronic Engineering, National Taipei Institute of Technology, Taipei, Taiwan, ROC. (E-mail: cct@en.tit.edu.tw) from source i to sink j, we define the slack s_{ij}:

$$s_{ij} = t_{ij} - a_{ij} \tag{1}$$

We define the slack s_{Bus} of the bus as the minimal slack s_{ij} between all possible pair of source i to sink j:

$$s_{Bus} = \min_{(i,j)} (s_{ij}) = \min_{(i,j)} (t_{ij} - a_{ij}) \tag{2}$$

To optimize the system performance, our goal is to maximize the slack s_{Bus} by buffer insertion and sizing.



(a)

Time interval	*Source*	*Sinks*
Period 1	p_1	p_3, p_6
Period 2	p_2	p_4, p_5
Period 3	p_1	p_2, p_3, p_5
Period 4	p_4	p_1, p_3, p_5, p_6

(b)

Fig. 1: An example of a bus with six terminals.

Fig. 2 shows a multiple-source bus with buffer insertion. There are buffers or bi-directional buffers on some segments of the bus. The direction of the signal flows are determined by arbitrators. We assume the wire widths in a bus is invariant, while the wire width of the control signals can be sized to match the timing requirements. Note that this is a distributed control system. For each source, the control signal is generated from the same block. The control signal triggers arbitrators which in turn sets the direction of the bi-directional buffers.

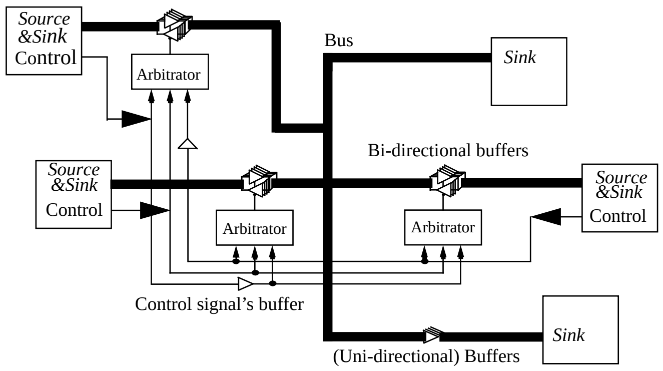


Fig. 2: A multiple-source bus with buffer insertion.

For the distributed control bus system (Fig. 2) we adopt the following assumptions:

(1) The potential buffer insertion locations are given.
(2) All the wire widths in a bus are homogeneous.
(3) The control signals arrive earlier than the data and meet the minimum setup time.

In this paper, we concentrate on buffer insertion and sizing. The general problem of bus buffer insertion can be stated as below:

Given a bus with N possible buffer locations and required arrival times t_{ij} of all source i to sink j pairs, identify the number of buffers, buffer insertion locations, and the buffer sizes, w_{bu}, for each buffer u, to maximize the slack s_{Bus} of the bus, with the constraint that $0 \leq w_{bu} \leq W_{max}$.

III. Delay Model

A. *Delay Between Active Components*

We choose the Elmore delay model [13] to estimate the signal delay. Fig. 3 shows the delay model of a segment (f,g) of wire, where l_{fg} is the length of wire (f,g), and R_w and C_w are the resistance and the capacitance per unit wire length, respectively. The resistance r_{fg} and capacitance c_{fg} of wire (f,g) are $r_{fg} = R_w l_{fg}$ and $c_{fg} = C_w l_{fg}$, respectively.

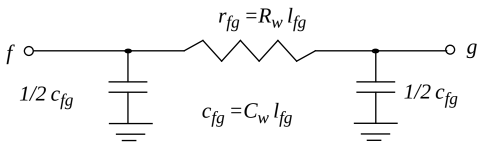


Fig. 3: Delay model of a bus wire.

For the delay from a buffer u to the next active component, we trace from the output of buffer u to construct a tree T_u with u as the root, and its descendent buffers/sinks as the leaves. Each branch of T_u corresponds to a wire segment and each internal node corresponds to a junction. For each leaf v, the path (u,v) from root u to v is unique. Based on the distributed RC tree of the Elmore delay model, the wire delay time from u to v, d_{uv}, can be denoted by

$$d_{uv} = \sum_{(f,g) \in path(u,v)} r_{fg} \cdot \left(\frac{1}{2} \cdot c_{fg} + c\left(T_g\right)\right) \tag{3}$$

where $c(T_g)$ is the lumped capacitance of the sub-tree rooted at the node g. The capacitance $c(T_g)$ can be partitioned into two terms, $c_w(T_g)$ the capacitance contributed by wires and $c_l(T_g)$ the capacitance contributed by the buffers and sinks, calculated by the following formula:

$$c(T_g) = c_w(T_g) + c_l(T_g) \tag{4}$$

where $c_w(T_g) = \sum_{h \in D(T_g)} (c_{gh} + c_w(T_h))$, (5)

$$c_l(T_g) = \begin{cases} C_b \cdot w_{bg} & \text{if } g \text{ is a buffer or sink,} \\ \sum_{h \in D(T_g)} c_l(T_h) & \text{otherwise} \end{cases} \tag{6}$$

where $D(T_g)$ is the set of children of node g.

We define r_{uvs} as the resistance of the common portion of the path between path (u,v) and path (u,s). The delay d_{uv} in equation (3) can be formulated as a function of buffer size w_{bs} at the leaves.

$$d_{uv} = \sum_{(f,g) \in path(u,v)} r_{fg} \cdot \left(\frac{1}{2} \cdot c_{fg} + c_w\left(T_g\right) + c_l\left(T_g\right)\right)$$

$$= \sum_{(f,g) \in path(u,v)} r_{fg} \cdot \left(\frac{1}{2} \cdot c_{fg} + c_w\left(T_g\right)\right) + \sum_{s \in S\left(T_u\right)} r_{uvs} C_b w_{bs}$$

$$= D_{0uv} + \sum_{s \in S(T_u)} r_{uvs} C_b w_{bs} \tag{7}$$

where $S(T_u)$ is the set of leaves in T_u, and

$$D_{0uv} = \sum_{(f,g) \in path(u,v)} r_{fg} \cdot \left(\frac{1}{2} \cdot c_{fg} + c_w(T_g)\right). \tag{8}$$

B. Bus Delay Model

A CMOS inverter can be a buffer where the output signal is the inverse of the input signal. For simplicity, two cascaded inverters, named the rear and the front inverters, are considered to be a buffer and inserted into a bus wire to maintain the signal polarity. For a bi-directional bus, we place two buffers in opposite directions and connect them together. Data can only pass in one direction within any timing period. This is achieved by using a tri-state inverter for the front inverter controlled by the control signal.

Fig. 4shows the symbol and the equivalent delay model for a buffer, where D_b, R_b, and C_b represent the intrinsic delay, the output driving resistance, and the input loading capacitance of a unit size inverter (*1*X), respectively. For a sized inverter (*w*X), the gate width increases by a factor of *w*, the output driving resistance is R_s/w, the input loading capacitance is $C_b w$; the intrinsic delay D_b is assumed to be a constant independent of *w*.

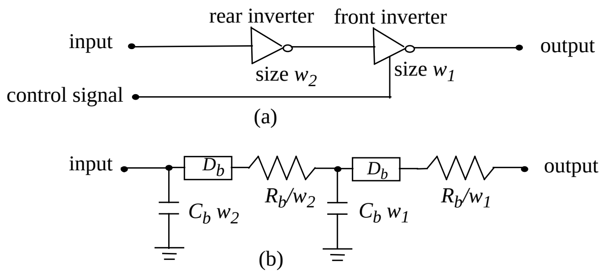


Fig. 4: (a) A buffer model and (b) the equivalent delay model.

With considering the buffer *u*, the delay d'_{uv} from buffer *u* to buffer *v* is:

$$d'_{uv} = D_b + \frac{R_b}{w_{bu}} c(T_u) + d_{uv} = D_b + \frac{R_b}{w_{bu}} \left(c_w(T_u) + c_l(T_u)\right) + d_{uv}$$

$$= D_b + \frac{R_b}{w_{bu}} \left(c_w(T_u) + \sum_{s \in S(T_u)} C_b w_{bs}\right) + d_{uv}$$

$$= D_b + \frac{1}{w_{bu}} \left(R_b c_w(T_u) + \sum_{s \in S(T_u)} R_b C_b w_{bs}\right) + D_{0uv} + \sum_{s \in S(T_u)} r_{uvs} C_b w_{bs}$$

$$= K_{0uv} + \frac{K_{1uv}}{w_{bu}} + \frac{R_b C_b}{w_{bu}} \sum_{s \in S(T_u)} w_{bs} + C_b \sum_{s \in S(T_u)} r_{uvs} w_{bs} \tag{9}$$

where $K_{0uv} = D_b + D_{0uv}$ and $K_{1uv} = R_b\ c_w(T_u)$. (10)

Given the source *i* and the sink *j*, let $(p_i, b_1, b_2,..., b_s, p_j)$ be the sequence of buffers along the path (p_i, p_j). We decompose the path (p_i, p_j) into a set B_{ij} of buffer (source or sink) pairs, *i.e.* $B_{ij} = \{(p_i, b_1), (b_1, b_2),..., (b_s, p_j)\}$.

With buffer insertion, the propagation delay based on the equation (9) from the source *i* to the sink *j* can be expressed as:

$$a_{ij} = \sum_{(u,v) \in B_{ij}} d'_{uv}$$

$$= \sum_{(u,v) \in B_{ij}} \left(K_{0uv} + \frac{K_{1uv}}{w_{bu}} + \frac{R_b C_b}{w_{bu}} \sum_{s \in S(T_u)} w_{bs} + C_b \sum_{s \in S(T_u)} r_{uvs} w_{bs}\right) \tag{11}$$

IV. Bus Buffer Insertion Algorithm

A. Overview

In the problem of bus buffer insertion, we attempt to insert buffers to maximize the slack s_{Bus}. We can formulate the problem in a nonlinear programing expression:

$$\text{Obj:} \quad \max s_{Bus} \tag{12}$$

Subject to the constraints derived from equations (1), (2), and (11):

$$\sum_{(u,v) \in B_{ij}} \left(K_{0uv} + \frac{K_{1uv}}{w_{bu}} + \frac{R_b C_b}{w_{bu}} \sum_{s \in S(T_u)} w_{bs} + C_b \sum_{s \in S(T_u)} r_{uvs} w_{bs}\right) + s_{Bus} \le t_{ij} \tag{13}$$

where $t_{ij} \geq 0$, $1 \leq i, j \leq n$, and $0 \leq w_{bu}, w_{bs} \leq W_{max}$.

Note that the coefficients K_{0uv}, K_{1uv}, R_b, C_b, and r_{uvs} in the left term of (13) are all nonnegative. The expressions are posynomial (positive polynomial). Let us set a variable z_i = ln w_{bi} for each buffer or sink. Posynomial can be transferred by an exponential transformation into convex functions. Thus, given the buffer placement, the sizing of the buffer is classified as a geometric programming problem [14]. Equation (13) can be rewritten as

$$\text{Obj:} \quad \max s_{Bus}$$

Subject to

$$\sum_{(u,v) \in B_{ij}} \left(K_{0uv} + K_{1uv} \cdot e^{-z_u} + R_b C_b \cdot e^{-z_u} \sum_{s \in S(T_u)} e^{z_s} + C_b \sum_{s \in S(T_u)} r_{uvs} e^{z_s}\right) \cdot t_{ij}^{-1} + s_{Bus} \cdot t_{ij}^{-1} \le 1 \tag{14}$$

The second derivatives of the second, third, and fourth terms in equation (14) with respect to z_u and z_s are positive. As a consequence, the equation (14) is a convex function of z_i. This convexity allows us to associate a stationary point uniquely with a minimum. Because the unique properties of z_i make it so suitable for optimization, we call z_i a natural variable. Since z_i is a monotonic function of w_{bi}, a stationary point with to z_i is, of course, also a stationary point with respect to w_{bi}. This unique property of z_i will make any locally optimal solution of (14) also globally optimal.

The second, third, and fourth terms in equation (13) are proportional to the buffer size w_{bs} at the root of each tree T_s but inversely proportional to the buffer size w_{bu} at the leaf, shown in Fig. 5. In the figure, equation (14) is convex with respect to ln w_{bi}. Fig. 6shows the outline of the combination of a set of equations (14). The solution space is convex.

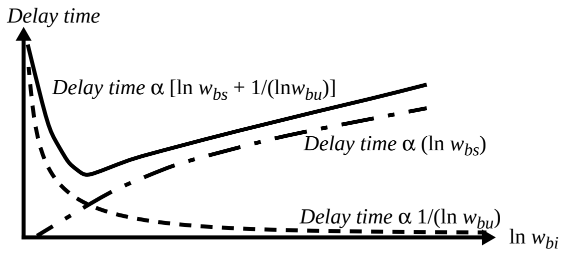


Fig. 5: Curve variation of the delay time a_{ij}

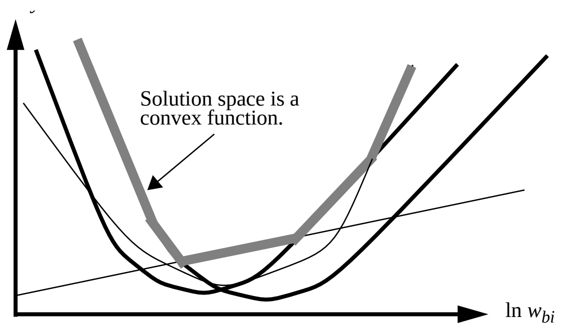


Fig. 6: The solution space of equation (14) is a convex.

Theorem 1:

The local optimal solution of obj (12) and constraints (14) is also globally optimal.

We propose a simple heuristic method to improve the bus buffer insertion iteratively. There are three levels of operations. At the top level, we try from $x = 1$ to N, to insert x buffers. The best result is selected. In the second level, for a given x, we search for the best buffer placement of the x buffers (Subsection IV.B). For each buffer location assignment, a third level of sizing operation is called to find the best buffer sizes such that the bus slack s_{Bus} is maximized (Subsection IV.C). The first level of the algorithm for *bus buffer insertion* is stated as below:

Bus_Buffer_Insertion ()

```
1 {  $s_{Bus} = -10^{-9} ns$;
2    For $x = 1$ to $N$, $x = x + 1$;
3    {  $cur\_slack$ = Buffer_Placement ($N$,$x$);
4       $s_{Bus} = \max(s_{Bus}, cur\_slack)$;
5    }
6 }
```

B. Buffer Placement

We solve the bus-buffer placement problem according to the combinatorial optimization approach proposed by Lin and Kernighan [12,15,16]. Given an initial assignment of x buffers and a number $0 < \lambda \le x$, in each iteration, we select λ buffers and try the placement of these λ buffers at all possible buffer locations. The placement that maximizes the bus slack s_{Bus} is kept. The entire process repeats until no further improvement is observed.

Lin and Kernighan define the solution to be a λ*-optimal*:

λ-optimal: A solution is λ*-optimal* if and only if the perturbation of any λ buffers on the buffer location does not improve the current solution.

To reduce the complexity of the operation, we set λ to be one. For each buffer placement, we optimize the buffer sizes. Therefore, the proposed buffer placement derives a 1-*optimal solution*.

The detailed procedure for the *buffer placement* is described below:

Buffer_Placement (*N*, *x*)

```
 1 {  Randomly place $x$ buffers into $N$ buffer locations;
 2    $s_{Bus} = -10^{9} ns$;
 3    Repeat
 4    {  For $i = 1$ to $x$, $i = i + 1$;
 5       {  $slack = 10^{9} ns$;
 6            For $j = 1$ to $(N-x)$, $j = j + 1$;
 7            {  Assign buffer $i$ to the $j$'th available buffer
                 location;
 8               $slack1$ = Buffer_Size_Decision ($x$);
 9               If $slack1 \le slack$
10               { $T = j$; $slack = slack1$ ;
11               }
12            } Assign buffer $i$ to $T$ buffer locations;
13       }
14    } until no improvement in $s_{Bus}$ is observed;
15    return ($s_{Bus}$);
16 }
```

Initially, the x buffers are randomly placed into N candidate locations. In each outer iteration (lines 3-14), we move each buffer to its best location until no further move can improve the slack. The inner iteration (lines 6-12) tries the N-x available buffer locations for a given buffer and returns the best move. Finally, the best result of the buffer placement is returned to the fist level.

C. Buffer Size Decision

Instead of using geometric programming, to simplify the implementation, we adopt a Gauss-Seidel iteration approach. Given a buffer placement, we adjust each buffer size w_{bu} to its optimal value assuming the rest of the buffer sizes are fixed. We use binary search based on a slope comparison for sizing the buffer size w_{bu} to get a maximum slack. The successive over-relaxation [17] is used to accelerate the convergence of the Gauss-Siedel iterations. Let w_{bu}^{*} be the optimal buffer size of w_{bu}, we have

$$w_{bu}^{k+1} = w_{bu}^{k} + \alpha\ (w_{bu}^{*} - w_{bu}^{k}) \qquad (15)$$

where k is the index of the iteration and α is the step size ($\alpha \ge 1$; experimentally, $\alpha = 1.2$).

The procedure of *buffer size decision* is described as follows:

```
Buffer_Size_Decision (x)
1 {  $k = 1$;
2    Repeat
3    {   For $u = 1$ to $x$, $u = u + 1$
4        {  $w_{bu}^{*}$ = Opt_Buffer (u);
5           $w_{bu}^{k+1} = w_{bu}^{k} + \alpha\ (w_{bu}^{*} - w_{bu}^{k})$;
6        }
7        $k = k + 1$;
8    } until $w_{bu}$ converges;
9    return ($S_{Bus}$);
10 }
```

For each buffer, we calculate the best buffer size with binary search (line 4). And then overshooting the buffer size (line 5) to accelerate the convergence. In practice, the front inverter should be sized before the rear inverter for faster convergence rate. Finally, the best results is returned to the second level, the buffer placement procedure.

D. Time Complexity

The bus buffer insertion algorithm consists of three hierarchical levels, the outer loop (Subsection IV.A), buffer placement (Subsection IV.B), and buffer size decision (Subsection IV.C). In the outer loop, it takes O(N) iterations since only one for-loop with $1 \leq x \leq N$ is called, where N is the number of buffer locations in a bus. In the procedure of the buffer placement, the number of iterations depends on the number of x buffers, $1 \leq x \leq N$. Since two hierarchical for-loops with the number of (N-x) and x are involved, the time complexity of the buffer placement is O(c_1N^2), where N is the number of buffer locations and c_1 (experimentally, $c_1 < 4$) is the number of repeat-loops. In the procedure of the buffer size decision, the for-loop takes O(N) and the subroutine of *Opt_buffer*() needs O($n \ln W_{max}$) since the binary search with maximum buffer size, W_{max}, is adopted. Thus the time complexity of the buffer size decision is O($c_2nN \ln W_{max}$), where c_2 (experimentally, $c_2 < 5$) is the number of repeat-loops.

In summary, the time complexity of the bus buffer insertion algorithm based on the combination of three hierarchical levels will be O($nN^4 \ln W_{max}$).

V. Experimental Results

The algorithm of bus-buffer insertion has been implemented in C language and runs on a PC-Pentium (60 Mhz) under MS-DOS 6.2. We adopt the CMOS technologies based on 2.0, 0.5, and 0.3 micron design rules [18]. For 0.5 micron technology, the input capacitance and output resistance of a unit size buffer are C_b=1.725fF and R_b=3170 Ohm, respectively. We suppose that the intrinsic delay is kept constant 230ps for any buffer sizes. For 0.3 micron technology, the input capacitance and output resistance of a unit size buffer are C_b=0.621fF and R_b=3170 Ohm respectively, and the intrinsic delay is kept constant 150ps for any buffer sizes. In addition, the wire resistance is 0.05 Ohm per square area and the wire capacitance is 0.1fF per micron. To represent a buffer sizing without size limitation, we set the maximum buffer size W_{max} to 250X. For a wire segment with bi-directional transmission, a bi-directional buffer is placed (if any) at the middle of the segment to balance the delay times contributed by the segment. But for a wire segment with uni-directional transmission, a buffer is placed at the end of the segment on the side of the sources. We also assume that all the sources driving capability can be adjusted but all the sinks have a fixed unit size loading.

Since there are no standard benchmarks available, test cases have been created and used to evaluate our algorithm. Table I summarizes the data of the test cases. The bus is assumed to reach the four edges of the chip core. The length of the critical path is measured along the path contributing the longest time delay. The number of locations is the sum of the number of the segments and sources. In all cases, the bus topologies are different and the required arrival times of all sinks are set to be 90% of the delay of critical path without buffer insertion. Case 1 has the simple source-sink pairs, Case 2 has the minimum die size, and Case 8 has more complicated bus structure.

TABLE I
The Data of Test Cases

Example	*Core Size* (μm x μm)	*Term Number*	*Critical Path* (μm)	*Source Number*	*Sink Number*	*Location Number*
Case 1	10000 x 14000	3	17000	2	2	5
Case 2	10000 x 5000	4	10000	3	3	9
Case 3	17000 x 10000	5	12000	3	3	10
Case 4	12900 x 7000	6	18900	3	5	17
Case 5	12000 x 10000	6	16000	4	4	16
Case 6	17000 x 15000	7	23000	7	6	18
Case 7	18000 x 15000	8	25000	5	4	17
Case 8	18000 x 14000	11	23000	9	7	26

Tables II, III, and IV show the results of both source driver sizing and bus buffer insertion based on 2.0, 0.5, and 0.3 micron technologies, respectively; where "*Delay*" is the maximum time delay from sources to sinks, "*Bsizes*" is the summation of all the inserted buffer sizes, and "*Cputime*" is the running time measured by Pentium-60MHz. From the tables, the "*Delay*" based on the buffer insertion approach is always less than that of the source driver sizing, but takes more buffer sizes. From the experiments, the average improvement in delay (*Delay saving*) is 7.2% for 2.0 micron technology, but 20.7% and 29.6% for 0.5 and 0.3 micron technologies, respectively at the expense of larger total buffer area.

TABLE II
Results of Both Driver Sizing and Buffer Insertion for 2.0 μm

Example	*Driver Sizing*			*Buffer Insertion*			*Delay Saving*
	Delay	*Bsizes*	*Cputime*	*Delay*	*Bsizes*	*Cputime*	
Case 1	3.973 ns	35	1 s	3.875 ns	48	4 s	2.5%
Case 2	3.579 ns	39	1 s	3.561 ns	40	16 s	0.5%
Case 3	4.448 ns	55	1 s	4.141 ns	66	44 s	6.9%
Case 4	4.411 ns	43	1 s	4.327 ns	59	399 s	1.9%
Case 5	5.480 ns	48	1 s	5.015 ns	76	333 s	8.4%
Case 6	7.926 ns	77	8 s	6.885 ns	126	778 s	13.1%
Case 7	7.335 ns	69	12 s	6.282 ns	167	1653 s	14.4%
Case 8	8.820 ns	96	84 s	7.960 ns	206	3796 s	9.8%
Average	-	-	-	-	-	-	7.2%

TABLE III
RESULTS OF BOTH DRIVER SIZING AND BUFFER INSERTION FOR 0.5 μm

Example	*Driver Sizing*			*Buffer Insertion*			*Delay Saving*
	Delay	*Bsizes*	*Cputime*	*Delay*	*Bsizes*	*Cputime*	
Case 1	2.190 ns	152	1 s	1.874 ns	257	3 s	14.4%
Case 2	1.751 ns	143	1 s	1.693 ns	155	36 s	3.3%
Case 3	2.560 ns	200	1 s	2.142 ns	333	190 s	16.3%
Case 4	2.484 ns	103	1 s	2.035 ns	263	873 s	18.1%
Case 5	3.298 ns	117	1 s	2.666 ns	349	814 s	19.2%
Case 6	4.866 ns	179	7 s	3.596 ns	515	1273 s	26.1%
Case 7	5.150 ns	232	5 s	3.486 ns	597	2835 s	32.3%
Case 8	5.988 ns	186	14 s	3.857 ns	516	10753 s	35.5%
Average	-	-	-	-	-	-	20.7%

TABLE IV
RESULTS OF BOTH DRIVER SIZING AND BUFFER INSERTION FOR 0.3 μm

Example	*Driver Sizing*			*Buffer Insertion*			*Delay Saving*
	Delay	*Bsizes*	*Cputime*	*Delay*	*Bsizes*	*Cputime*	
Case 1	1.770 ns	285	1 s	1.439 ns	493	4 s	18.7%
Case 2	1.339 ns	251	1 s	1.286 ns	276	55 s	4.0%
Case 3	2.131 ns	312	1 s	1.601 ns	822	254 s	24.9%
Case 4	2.125 ns	128	2 s	1.583 ns	507	1243 s	25.5%
Case 5	2.767 ns	202	1 s	1.928 ns	669	1413 s	30.3%
Case 6	4.292 ns	349	10 s	2.718 ns	866	1677 s	36.7%
Case 7	4.727 ns	544	9 s	2.365 ns	1324	3783 s	50.0%
Case 8	5.114 ns	452	21 s	2.728 ns	1117	13940 s	46.7%
Average	-	-	-	-	-	-	29.6%

We use the same topology but different bus length to compare the impact of core size on the bus buffer insertion. Fig. 7 shows the comparison for Case 5 for 0.5 μm technology. The x-axis represents the percentage of the core size with respect to the original core size. From the figure, we observe that the delay time with the buffer insertion is reduced as the core size shrinks.

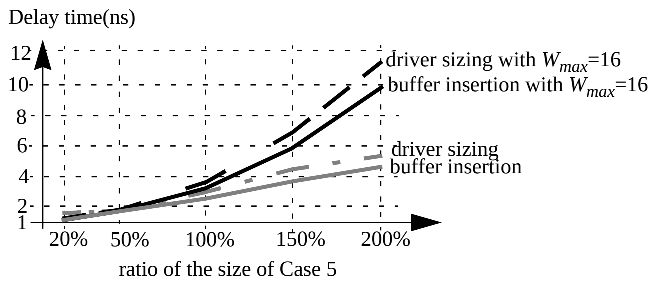


Fig. 7: Variations of the different core sizes of Case 5 with/without size limitation in delay and buffer size for 0.5 μm technology.

VI. CONCLUSION

The bus buffer insertion algorithm has been proposed and implemented. The algorithm is designed to minimize the clock period by inserting buffers in the multi-source and multi-sink bus. As wire resistance increases with larger chip sizes and finer wire widths, bus buffer insertion yields greater improvements in performance. In our experiments, the improvements increase from 7.6% to 20.7% and 29.6% when we move the technology from 2.0 microns to 0.5 and 0.3 micron technologies.

Besides delay reduction, buffer insertion can remedy noise and cross talk problems. Bus buffer may restore the signal before the noise corrupts the data. The inverting buffer can be used to change the phase of signals and thus reducing cross talk. Future work includes area limitation and power consumption.

REFERENCES

[1] J. P. Fishburn and A. E. Dunlop, "TILOS: a posynomial programming approach to transistor sizing," *IEEE International Conference on Computer-Aided Design,* pp. 326-328, 1985.

[2] P. K. Chan, "Algorithm for library-specific sizing of combinational logic," *Proc. of 27th ACM/IEEE Design Automation Conference,* pp. 353-356, 1990.

[3] M. R. C. M. Berkelaar and J. A. G. Jess, "Gate sizing in MOS digital circuits with linear programming," *Proc. of European Design Automation Conference,* pp. 217-221, 1990.

[4] A. C. H. Wu, N. Vander Zanden, and D. Gajski, "A new algorithm for transistor sizing in CMOS circuit," *Proc. of European Design Automation Conference,* pp. 589-593, 1990.

[5] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang, "An exact solution to the transistor sizing problem for CMOS circuits using convex optimization," *IEEE Trans. on Computer Aided Design,* Vol. 12, No. 11, 1993, pp. 1621-1634.

[6] S. Mehrotra, P. Franzin, and W. Liu, "Stochastic optimization approach to transistor sizing for CMOS VLSI circuits," *Proc. of 31th ACM/IEEE Design Automation Conference,* pp. 36-40, 1994.

[7] L. V. Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," *Proc. of International Symposium on Circuits and Systems,* pp. 865-868, 1990.

[8] J. D. Cho and M. Sarrafzadeh, "A buffer distribution algorithm for high-speed clock routing," *Proc. of 30th ACM/IEEE Design Automation Conference,* pp. 537-543, 1993.

[9] S. Pullela, N. Menezes, J. Omar, L. Pillage, "Skew and delay optimization for reliable buffered clock trees," *Proc. of IEEE International Conference on Computer-Aided Design,* pp. 556-562, 1993.

[10] J. Chung and C. K. Cheng, "Skew sensitivity minimization of buffered clock tree," *Proc. of IEEE International Conference on Computer-Aided Design,* pp. 280-283, 1994.

[11] S. Lin and B. W. Kernighan, "Heuristic solution of a concentrator location problem," *Bell Lab. Technical Memorandum*, 1976.

[12] S. Lin, "Effective use of heuristic algorithms in network design," *Proc. of Symposia in Applied Mathematics,* Vol. 26, pp. 63-84, 1982.

[13] W. C. Elmore, "The transient response of damped linear networks with particular regard to wide-band amplifiers," *Journal of Applied Physicals,* 19(1), pp. 55-63, Jan. 1948.

[14] C. S. Beightler and D. T. Phillips, *Applied geometric programming,* Wiley, New York, 1976.

[15] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," *Operations Research*, 21 (2), pp. 498-516, 1973.

[16] S. Lin, "Heuristic programming as an aid to network design," *Networks* 5 (1), pp. 33-45, Jan. 1975.

[17] Sergio Pissanetsky, *Sparse Matrix Technology,* Academic Press Inc. LTD., London, 1984.

[18] *NCR ASIC Data Book,* NCR Corporation, Dayton, Ohio, USA, Jan. 1987.