A New RLC Buffer Insertion Algorithm

Zhanyuan Jiang, Shiyan Hu, Jiang Hu, Zhuo Li† and Weiping Shi
Texas A&M University, College Station, Texas 77843
†IBM Austin Research Lab, Austin, Texas 78758
{jerryjiang, hushiyan, jianghu, wshi}@ece.tamu.edu
†lizhuo@us.ibm.com

ABSTRACT

Most existing buffering algorithms neglect the impact of inductance on circuit performance, which causes large error in circuit analysis and optimization. Even for the approaches considering inductance effects, their delay models are too simplistic to catch the actual performance. As delay-length dependence is approaching linear with inductance effect [1], fewer buffers are needed to reduce RLC delay. This motivates this work to propose a new algorithm for RLC buffer insertion.

In this paper, a new buffer insertion algorithm considering inductance for intermediate and global interconnect is proposed, based on downstream impedance instead of traditional downstream capacitance. A new pruning technique that provides tremendous speedup and a new frequency estimation method that is very accurate in delay computation are also proposed.

Experiments on industrial netlists demonstrate that our new algorithm reduces the number of buffers up to 34.4% over the traditional van Ginneken's algorithm that ignores inductance. Our impedance delay estimation is very accurate compared to SPICE simulations, with only 10% error while the delay model used in the previous RLC algorithm has 20% error [2]. The accurate delay model not only reduces the number of buffers, but also brings high fidelity to the buffer solutions. Incorporating slew constraints, the algorithm is accelerated by about 4× with only slight degradation in solution quality.

1. INTRODUCTION

With higher operating frequencies, increasing concerns in the effect of on-chip inductance have been raised [3]. Compared to an accurate RLC model, RC model can create 60% timing error in the current copper interconnect technologies [4]. On the other hand, as inductance effects aggravate, the quadratic delay-length dependence in RC model is approaching linear [1], which may result in significant buffer savings. Thus, the inductance has a significant impact on timing analysis and optimization for interconnect.

In this paper, we propose a new RLC buffering algorithm based on an accurate RLC model. The new algorithm uses dynamic programming framework to provide near-optimal performance, in contrast to [2] which has the trial-and-error flavor as a greedy algorithm. Main features of the new algorithm are summarized as follows.

- Based on downstream impedance, a new buffering formulation is proposed to handle RLC interconnect. This improves the widely-used downstream capacitance or moment based formulations [5].
- Based on new properties of the RLC model, an effective pruning technique is proposed to speed up the algorithm. The properties include the fact that the delay of a buffer decreases with the real part of its downstream impedance, and increases with the imaginary part.
- Constraint on slew rate is handled. Incorporating slew constraint provides further speedup and makes our work ready for practical use.

The rest of the paper is organized as follows: Section 2 describes the delay model. Section 3 describes the proposed RLC buffering algorithm. Section 4 presents the experimental results with analysis. A summary of work is given in Section 5.

2. DELAY MODEL

Since the prevailing delay model is too simplistic to catch the actual performance considering inductance effect, 60% timing error is often observed [4]. Thus, more accurate delay models are necessary for accurate timing analysis and buffer insertion. Such a model will be introduced in this section.

2.1 Realistic Range of Inductance

We set up a realistic environment to extract the parasitics based on the model of [6]. FastCap and FastHenry are used to extract capacitance and inductance. For MOSIS $130nm$ technology [7], the following parameters are obtained. From metal layer one to six, unit resistance varies from $350\Omega/mm$ to $10\Omega/mm$, unit capacitance from $380fF/mm$ to $180fF/mm$ and unit inductance from $0.6nH/mm$ to $1.3nH/mm$.

*This research was supported in part by SRC grant 2004-TJ-1205, NSF grant EIA-0223785, and IBM faculty research award.

ICCAD'06, November 5-9, 2006, San Jose, CA

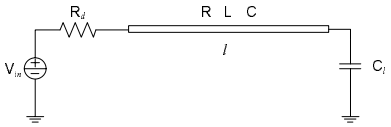

Figure 1: A single transmission line [8].

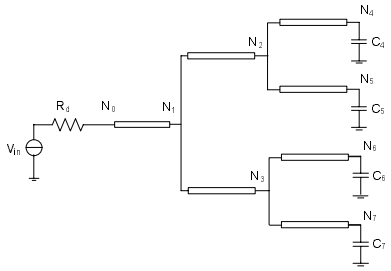

Figure 2: An RLC tree.

It is worth mentioning that [2] handles RLC buffering based on the model in [1]. Although the theory in [1] is solid, the unit impedance value in [1] is large. For example, the range of unit inductance there is $10nH/mm$ to $1000nH/mm$ which can be computed using Table 1 in [1]. Our work uses more realistic unit inductance value.

2.2 Interconnect Model

A transmission line model with a driver and a load is shown in Figure 1, which is also used in [8]. There, the driver is connected to an input voltage source, l denotes the interconnect length, and R, L, C denote unit resistance, inductance and capacitance, respectively. The driver is modeled as a resistor R_d and the load is modeled as a capacitor C_l.

In this paper, an accurate RLC interconnect model from [9] is employed. The transfer function from the input to the output of a transmission line is

$$H(s) = \frac{1}{(1 + R_d C_l s)\cosh\theta + (R_d/Z_c + Z_c C_l s)\sinh\theta}, \quad (1)$$

where $\theta = l\sqrt{(R+sL)/sC}$ and $Z_c = \sqrt{(R+sL)/sC}$. It is demonstrated in [8] that the timing analysis based on this model is on average only 3% off SPICE simulation results.

For a general RLC tree shown in Figure 2, the transfer function from N_0 to an internal node N_i is the product of the transfer functions of all branches along the path from N_0 to N_i [8]:

$$H(s) = \frac{Z_{L,0}}{R_d + Z_{L,0}} \prod_k \frac{1}{\cosh\theta_k + (Z_{c,k}/Z_{L,k})\sinh\theta_k}, \quad (2)$$

where $Z_{L,0}$ is the input impedance seen from N_0 and k is the index of branches in the path from N_0 to N_i.

For a transmission line of length l with load Z_L, the input impedance is:

$$Z_{down} = Z_C \frac{Z_L + Z_C \tanh\theta}{Z_C + Z_L \tanh\theta}, \quad (3)$$

where Z_C and θ is defined in Eqn. (1).

2.3 Signal and Buffer Modeling

Both input signal and output signal are modeled as a DC component and two harmonics. The input signal is given by

$$V_{in}(t) = \frac{V_{dd}}{2} + A_1 \sin(\omega_1 t + \phi_1) + A_2 \sin(\omega_2 t + \phi_2), \quad (4)$$

where A, ω, ϕ are the magnitudes, angular frequencies, and phases of sinusoidal signals, respectively. A ramp signal is adopted to illustrate the effectiveness of the above model. As is well known, a ramp signal with a transition time τ is given by

$$V_{in}(t) = \begin{cases} \frac{t}{\tau} V_{dd} & 0 \le t < \tau, \\ V_{dd} & \tau \le t. \end{cases} \quad (5)$$

Since only the whole transient state and part of steady state of signals contribute to non-zero frequency components of following stages, we can chop the signal to keep the first 3-10τ duration of signal, which contains dominant components of signal in frequency domain. Least square method is applied to approximate the chopped input signal:

$$\min \sum_{i=1}^{N} (y_i - (\frac{V_{dd}}{2} + A_1 \sin(\omega_1 t_i + \phi_1) + A_2 \sin(\omega_2 t_i + \phi_2)))^2,$$

where N is the number of sampling points and the unknown vector is $(A_1, \omega_1, \phi_1, A_2, \omega_2, \phi_2)$. To simplify the successive tasks, we set the angular frequency of the second harmonic to be 3 times that of the first harmonic, namely, $\omega_2 = 3\omega_1$. Our later experiments also verify that such approximation is enough for the ramp input.

The above model can be used to model both input and output of a non-linear driver model. For a slew input, least square method is applied to find basic and third-order frequency. For each buffer type, since the output signal depends on both the input signal waveform and downstream capacitance, we perform SPICE simulation to cover all parameter ranges and for each output signal, least square method is applied to find the basic and third-order harmonics. A two dimensional look-up table is constructed for each buffer type accordingly, where basic frequency of output signal is searched through basic frequency of input signal and downstream capacitance. With such an approach, we can approximate all signals with a bunch of frequency bins.

2.4 Impedance Delay

Computation of output delay is adopted from [8]. The procedure is omitted here due to space limit. In this papar, a new term *impedance delay* is introduced to characterize the delay due to impedance.

Table 1: Comparison between *impedance delay* and SPICE. The interconnect is modeled as 200 unit length segments using π model.

L(μm)	$R_d(\Omega)$	$C_l(pf)$	Impe.(ps)	SPICE(ps)	Error
500	100	50	17.8	17.5	1.7%
500	100	100	23.4	23.0	1.7%
500	500	50	53.2	51.9	2.4%
500	500	100	72.7	70.3	3.3%
1000	100	50	32.2	31.4	2.5%
1000	100	100	38.2	37.0	3.1%
1000	500	50	92.8	87.9	5.3%
1000	500	100	113.0	106.1	6.1%

Experiments are performed to test the accuracy of the method. The transition time τ of the input signal is set to $50ps$ and the duration of chopped signal is set to 5τ. Refer to Table 1 and Figure 3 for the results. A wide range of circuit parameters are applied and the maximum timing error is only 6.1%.

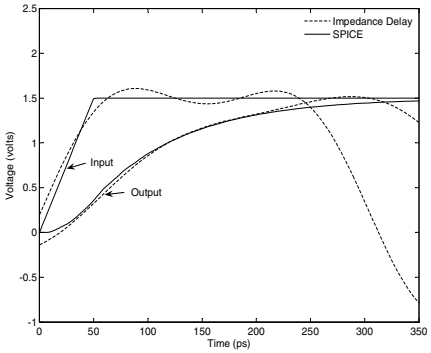

Figure 3: Comparison of input and output signals between *impedance delay* and SPICE. C_l**=50**fF**,** R_d**=580**Ω **and Length= 500**μm**.**

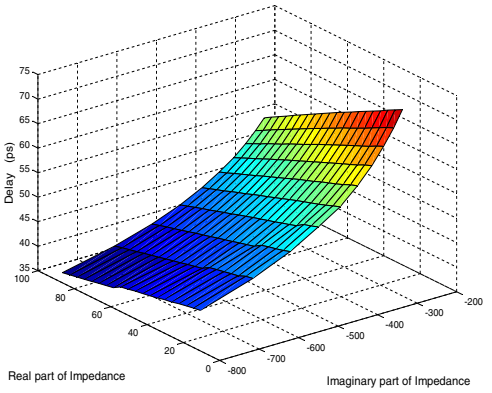

Figure 4: Buffer delay trend with varying downstream impedence.

3. RLC BUFFERING ALGORITHM

3.1 Preliminaries

The basic buffering problem includes a routing tree $T = (V, E)$, where $V = \{s_0\} \cup V_s \cup V_n$, and $E \subseteq V \times V$. Vertex s_0 is the *source* vertex, V_s is the set of *sink* vertices and V_n is the set of *internal* vertices. Each sink vertex $s \in V_s$ is associated with sink capacitance C_s, and each edge $e \in E$ is associated with lumped resistance R_e and capacitance C_e. A buffer library B contains different types of buffers. Each type of buffer b has a cost W_b, which can be measured by area or any other metric, depending on the optimization objective. Without loss of generality, we assume that the driver at source s_0 is also a buffer. A function $f : V_n \rightarrow 2^B$ specifies the types of buffers allowed at each internal vertex. A buffer assignment γ is a mapping $\gamma : V_n \rightarrow B \cup \{\wedge\}$ where $\wedge$ denotes that no buffer is inserted. The cost of a solution γ is $W(\gamma) = \sum_{b \in \gamma} W_b$. With the above notations, our RLC buffering problem can be formulated as follows.

RLC Minimum Cost Buffer Insertion with Slew Constraint Problem: Given a routing tree $T = (V, E)$, possible buffer positions defined by f, and a buffer library B, find a buffer assignment γ such that the total cost $W(\gamma)$ is minimized, the RLC require arrival time at the driver is no less than a given constant α and the input slew at each buffer is no greater than a given constant β.

3.2 New Pruning Condition

The new RLC algorithm works under the dynamic programming framework but using the impedance delay model. In order to handle impedance, solution characterization and pruning conditions need to be modified. These are described as follows.

3.2.1 Handling Impedance

Unlike capacitance, impedance depends on frequency. Since an input signal is expressed using basic and third frequency information, our buffering algorithm needs to consider both frequencies. Speedup techniques are necessary to obtain an efficient algorithm. The speedup is based on the following critical observations.

Observation 1 When adding a buffer/wire to drive an RLC network, the delay of a buffer/wire decreases with the real part of its downstream impedance and increases with the imaginary part of its downstream impedance.

To validated this observation, we perform extensive SPICE simulations on single transmission lines and general trees to guarantee the accurate results. For a single transmission line, we model the downstream interconnect line as 200 segments of unit length and each segment is modeled using π model. First, we fix unit inductance to $1nH/mm$ and vary unit resistance and unit capacitance. Figure 4 shows the trend of the delay after adding a buffer due to different downstream impedances. The trend of the delay after adding a wire is similar. We then vary unit inductance from $0.6nH/mm$ to $1.3nH/mm$, the trend remains. Cases with different downstream lengths and input frequencies are also investigated. For a general tree, the similar experiments are performed and the same trend is observed.

Observation 2 The impedance at third-order frequency has much less impact on delay than the impedance at basic frequency.

This is straightforward as higher-order frequency shows less impact on the output delay. In our case, since the magnitude of the signal at third-order frequency is around 1/9 of that at the basic frequency, the impedance at third-order frequency has much less impact on the output delay.

Although the impedance at third-order frequency is useful for performing an accurate timing analysis on the circuit, it is much less useful for comparing solutions. As a consequence of *Observation 2*, timing comparison between two solutions remains the same even when the impedance at third-order frequency is dropped. As such, only impedance at basic frequency needs to be compared for domination check and impedance at third-order frequency is just used for delay evaluation. This allows a tremendous speedup over the consideration of impedances at both basic and third-order frequencies.

To handle impedance, the downstream impedance Z is introduced to replace C in van Ginneken's algorithm. Since Z is a complex number consisting of a real part and imaginary part, denoted by Z_r, Z_i, respectively. Denote by Z_{1r}, Z_{1i} the impedance at basic frequency and by Z_{3r}, Z_{3i} the impedance at third-order frequency.

3.2.2 Pruning Conditions

The new pruning condition goes as follows. For any two solutions γ_1, γ_2 *at the same node*, γ_1 *dominates* γ_2 if $Q(\gamma_1) \geq Q(\gamma_2)$, $Z_{1r}(\gamma_1) \geq Z_{1r}(\gamma_2)$, $Z_{1i}(\gamma_1) \leq Z_{1i}(\gamma_i)$ and $W(\gamma_1) \leq W(\gamma_2)$. Whenever a solution becomes dominated, it is pruned from the solution set without further propagation.

3.3 Algorithm

Our algorithm shares the same dynamic programming framework as van Ginneken's algorithm, but has critical underlying differences. The differences include handling impedance, frequency bin and slew constraint.

In the dynamic programming framework, a set of candidate solutions are propagated from the sinks toward the source along the given tree. Each solution γ is characterized by tuples. The first tuple is $(Q(\omega), Z_{1r}(\omega), Z_{1i}(\omega), W)$, which is used in domination check/pruning as mentioned in Section 3.2.2. Note that ω is involved since delay and impedance depend on the frequency ω. As frequency bin is used in this paper, ω corresponds to the average frequency in a frequency bin. The second tuple is $(Z_{3r}(\omega), Z_{3i}(\omega))$, which is used for accurately calculating delay but not pruning. The third tuple is $(S(\omega), C)$, which is only responsible for eliminating infeasible solutions. Refer to [10] for slew computation. Once again, although there are many tuples in the algorithm, only the first tuple is used for domination check and pruning dominated solutions. Thus, our algorithm is still efficient.

The procedure of the new buffering algorithm is as follows. At a sink node, Q is equal to the required arrival time at that sink, $Z_{1r}(\omega) = 0$, $Z_{3r}(\omega) = 0$, $Z_{3r}(\omega) = -\frac{1}{C\omega}$, $Z_{3i}(\omega) = -\frac{1}{3C\omega}$, $W = 0$ and $S(\omega) = 0$, where C is sink capacitance and ω represents a frequency bin.

Consider to propagate solutions from a node v to its parent node u through edge $e = (u, v)$. A solution γ_v at v becomes solution γ_u at u, $Z_{1r}(\omega), Z_{1i}(\omega), Z_{3r}(\omega), Z_{3i}(\omega)$ can be calculated by Eqn. (3), $C(\gamma_u) = C(\gamma_v) + C_e$, $W(\gamma_u) = W(\gamma_v)$. $Q(\omega)$ is not updated during wire insertion and updating is carried out when performing buffer insertion and branch merge.

In addition to keeping the unbuffered solution γ_u which is corresponding to a certain frequency ω, a buffer b_i can be inserted at u to generate a buffered solution $\gamma_{u,buf}$ which can be then computed as $Q(\gamma_{u,buf}) = Q(\gamma_u) - D(\gamma_{down}) - K_{b_i}$, where $D(\gamma_{down})$ is the total downstream *impedance delay* computed in Section 2.4 from the node u to its child node (which can be a sink or buffer). Note that $Q(\gamma_{u,buf})$ may correspond to a frequency other than γ_u due to buffer insertion. $D_{\gamma_{down}}$ is computed using delay re-evaluation. This is necessary as our delay model is not additive. After buffer insertion, $C(\gamma_{u,buf}) = C_{b_i}$, $Z_{1r}(\omega) = 0$, $Z_{3r}(\omega) = 0$, $Z_{3r}(\omega) = -\frac{1}{C_{b_i}\omega}$, $Z_{3i}(\omega) = -\frac{1}{3C_{b_i}\omega}$, $W(\gamma_{u,buf}) = W(\gamma_v) + W_{b_i}$ and $S(\gamma_{u,buf,\omega}) = 0$.

Denote the left-branch solution set and the right-branch solution set by $\Gamma_l(\omega)$ and $\Gamma_r(\omega)$, respectively. Since the signal frequencies of left child branch and right child branch are always the same, only solutions at the same frequency bin are merged. For each solution $\gamma_l(\omega) \in \Gamma_l(\omega)$ and each solution $\gamma_r(\omega) \in \Gamma_r(\omega)$, the corresponding merged solution $\gamma'(\omega)$ can be obtained according to $Q(\gamma'(\omega)) = \min\{Q(\gamma_l(\omega)), Q(\gamma_r(\omega))\}$. Each Z at its frequency can be merged by the rule of calculating parallel impedance,

$$Z(\gamma'(\omega)) = \frac{Z(\gamma_l(\omega))Z(\gamma_r(\omega))}{Z(\gamma_l(\omega)) + Z(\gamma_r(\omega))}. \quad (6)$$

$C(\gamma') = C(\gamma_l)+C(\gamma_r)$, $W(\gamma') = W(\gamma_l)+W(\gamma_r)$ and $S(\gamma'(\omega)) = \max\{S(\gamma_l(\omega)), S(\gamma_r(\omega))\}$.

4. EXPERIMENTAL RESULTS

4.1 Experiments Setup

All algorithms are implemented in C++ and are tested on a Pentium IV computer with a 3.2GHz CPU and 1GB memory. Our test cases are extracted from an industrial ASIC chip, which consists of 1000 nets with more than 50000 nodes including sinks, branching nodes and buffer positions. Among them, 682 nets have ≤ 5 sinks and all the remaining nets have ≤ 20 sinks. The sink capacitances range from $2.5fF$ to $200fF$. The unit resistance is $16.42\Omega/mm$, the unit capacitance is $194.2fF/mm$ and the unit inductance is $1.017nH/mm$. The buffer library consists of 12 buffers, in which 7 are non-inverting and 5 are inverting . Buffer slew resistances range from 60Ω to 730Ω and input capacitances range from $2.1fF$ to $76.0fF$. The range of input frequency bin is from 1GHz to 3GHz, we discretize the input signal into 5 frequency bins and the downstream capacitance into 10 capacitance bins. The time unit for this section is *ps* if not specified. SPICE simulation is based on RLC model in all the experiments below.

For convenience, all algorithms in comparison are listed below together with their abbreviations.

- VGL: van Ginneken/Lillis's min-cost timing buffering based on the Elmore delay.
- NEW: new RLC min-cost timing buffering algorithm based on *impedance delay*.
- NEW+S: new RLC min-cost timing buffering algorithm with slew constraint.

4.2 The Optimality of NEW Algorithm

Forty small testcases each having a dozen candidate buffer positions are used to verify the optimality of our algorithm. The testcases include 20 balanced trees and 20 unbalanced trees. For simplicity, only a single buffer type is used. Since all trees are very small, we can verify the optimality of our algorithm through exhaustive search based on SPICE. To this end, we enumerate all possible buffering solutions and compare the best solution there with the one by our algorithm. SPICE simulation is used for timing analysis. Compared to the best solution by exhaustive search, the new algorithm inserts the same number of buffers at the same positions in 16 balanced trees and 13 unbalanced trees, and adds/misplaces one or two buffers in the other trees. All of our timing analysis results are close to SPICE simulations. The results of eight example trees are summarized in Table 2, where B1, B2, B3, B4 refer to balanced trees and UB1, UB2, UB3, UB4 refer to unbalanced trees.

Table 2: Comparison between VGL, NEW and Exhaustive SPICE Search.

	VGL		NEW		SPICE (Exhaustive)	
Test Cases	Buffered Delay	# Buf.	Buffered Delay	# Buf.	Buffered Delay	# Buf.
B1	126	2	114	1	114	1
B2	161	3	151	2	151	2
B3	113	4	95	3	95	3
B4	110	4	100	3	92	2
UB1	141	2	115	1	115	1
UB2	307	5	280	3	273	3
UB3	404	4	372	1	372	1
UB4	332	5	322	4	310	3

Table 3: Comparison between VGL and NEW on five sets of testcases, each having 200 nets.

	VGL			NEW				
Test Sets	Avg. De.	# Buf.	CPU (s)	Avg. De.	# Buf.	CPU (s)	SP-ICE	Buf. Saving
S1	2072	1722	465	1664	1130	1430	1502	34.4 %
S2	1983	2207	510	1432	1552	1357	1289	29.7 %
S3	1656	1929	502	1306	1332	1364	1172	30.9 %
S4	1591	1593	448	1239	1256	1302	1155	21.2 %
S5	1488	1579	430	1192	1209	1293	1079	23.4 %

4.3 Comparison between VGL and NEW

We compare buffer reduction and CPU time between VGL and NEW algorithms. The results on totally 1000 nets are summarized in Table 3 where "buffer saving" refers to percentage difference in the number of buffers. We make the following observations:

- The new algorithm saves up to 34.4% buffers over VGL algorithm. One reason for huge buffer saving is that the delay of NEW is approaching linear with inductance effect. Another reason is that delay is overestimated using the traditional Elmore delay model and thus excessive buffers are inserted.
- SPICE simulation is performed on the circuits resulting from NEW algorithm. One can see that the average timing error is only 10%.

4.4 Comparison between VGL and NEW under timing constraints

We carry out the algorithms on 100 nets whose results are summarized in Table 4. At the driver, the solution satisfying the timing constraint with minimum number of buffers is chosen. Again, delay evaluation uses SPICE. From Table 4, we make the following observations:

- NEW algorithm always gives better solutions than VGL algorithm. For the same delay, NEW algorithm uses less number of buffers.
- NEW can reduce up to 26.1% of buffers for different timing constraint.
- Tight timing constraint may cause VGL to return no feasible solutions while NEW is still able to return feasible solutions.

Table 4: Comparison between VGL and NEW under different timing constraints.

Timing Constraint	#Buf. VGL	#Buf. NEW	Buffer Saving
1800	1606	1187	26.1%
2000	1114	863	22.5%
2200	825	676	18.1%
2400	491	414	15.6%
2600	379	343	9.5%

4.5 Comparison between NEW and NEW+S

Results are summarized in Table 5. Comparing to Table 3, we observe the following:

- The number of buffers decreases for NEW+S as the slew constraint loosens. This makes sense since a looser constraint means that buffers can be spaced further apart.
- NEW+S is up to 4× faster than the NEW algorithm, in spite of considering all the 12 buffers in the buffer library. A lot of infeasible solutions have been pruned due to slew constraint violation. For example, when we set the slew constraint to $300ps$, there are only 5 solutions on average for each frequency bin of each net, while the number is 60 without slew constraint.
- Considering slew, one sees that NEW+S only slightly sacrifices delay. This demonstrates that slew buffering is effective to speed up the algorithm.

Table 5: Comparison of buffering between NEW and NEW+S on five sets of testcases. Each set consists of 200 nets.

	Slew=$300ps$			Slew=$500ps$		
Test Sets	Avg. Delay	# Buf.	CPU (s)	Avg. Delay	# Buf.	CPU (s)
S1	1715	1492	482	1681	1339	873
S2	1465	1887	367	1442	1734	797
S3	1323	1649	371	1314	1466	805
S4	1258	1546	359	1242	1433	788
S5	1223	1453	310	1217	1360	725

5. CONCLUSION

This work proposes a new buffering algorithm considering inductance effects to meet the demand of accurate timing optimization in high frequency digital circuits. The new algorithm proposes an efficient yet accurate buffering formulation to estimate inductance effects and guide the solution pruning. The algorithm is also extended to include buffer cost and slew constraints for practical use. Experiments on industrial netlists show that buffer insertion considering inductance effects can save up to 34.4% of the resources while giving more accurate timing.

6. REFERENCES

[1] Y.I. Ismail and E.G. Friedman, "Effects of inductance on the propagation delay and repeater insertion in VLSI circuits," *DAC*, pp. 721–724, 1999.

[2] Y.I. Ismail, E.G. Friedman and J.L. Neves, "Repeater insertion in tree structured inductive interconnect," *ICCAD*, pp. 420 – 424, 2001.

[3] Y. Ismail, E. Friedman, and J. Neves, "Figures of merit to characterize the importance of on-chip inductance," *DAC*, pp. 560–565, 1998.

[4] Y.I. Ismail and E.G. Friedman, "Effects of inductance on the propagation delay and repeater insertion in VLSI circuit: a summary," *IEEE Circuits and Systems Magazine*, vol. 3, no. 1, pp. 24–28, 2003.

[5] C.J. Alpert, A. Devgan and S.T. Quay, "Buffer insertion with accurate gate and interconnect delay computation," *DAC*, pp. 479–484, 1999.

[6] K. Gala, V. Zolotov, R. Panda, B. Young, J. Wang, and D. Blaauw, "On-Chip inductance modeling and analysis," *DAC*, pp. 63–68, 2000.

[7] "http://www.mosis.com."

[8] G. Chen and E. Friedman, "An RLC interconnect model based on fourier analysis," *TCAD*, vol. 24, no. 2, pp. 170 – 183, 2005.

[9] L.N. Dworsky, *Modern Transmission Line Theory and Applications*, 1979.

[10] S. Hu, C.J. Alpert, J. Hu, S. Karandikar, Z. Li, W. Shi and C.-N. Sze, "Fast algorithms for slew constrained minimum cost buffering," *DAC*, pp. 308–313, 2006.