Microelectronics Journal 40 (2009) 1482–1486

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

A fast general slew constrained minimum cost buffering algorithm[☆]

Shiyan Hu [a,*], Jiang Hu [b]

[a] Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931, USA
[b] Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843, USA

ARTICLE INFO

Article history:
Received 22 December 2008
Accepted 10 August 2009

Keywords:
Buffer insertion
Slew constraint
Non-fixed input slew
Interconnect optimization
Physical design

ABSTRACT

As VLSI technology moves to the nanoscale regime, ultra-fast slew buffering techniques considering buffer cost minimization are highly desirable. The existing technique proposed in [S. Hu, C. Alpert, J. Hu, S. Karandikar, Z. Li, W. Shi, C.-N. Sze, Fast algorithms for slew constrained minimum cost buffering, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (11) (2007) 2009–2022] is able to efficiently perform buffer insertion with a simplified assumption on buffer input slew. However, when handling more general cases without input slew assumptions, it becomes slow despite the significant buffer savings. In this paper, a fast buffering technique is proposed to handle the general slew buffering problem. Instead of building solutions from scratch, the new technique efficiently optimizes buffering solutions obtained with the fixed input slew assumption. Experiments on industrial nets demonstrate that our algorithm is highly efficient. Compared to the commonly used van Ginneken style buffering, up to 49× speed up is obtained and often 10% buffer area is saved. Compared to the algorithm without input slew assumption proposed in [S. Hu, C. Alpert, J. Hu, S. Karandikar, Z. Li, W. Shi, C.-N. Sze, Fast algorithms for slew constrained minimum cost buffering, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (11) (2007) 2009–2022], up to 37× speedup can be obtained with slight sacrifice in solution quality.

1. Introduction

As VLSI technology moves to the nanoscale regime, devices scale much faster than interconnects. As a highly effective interconnect optimization technique, buffer insertion is extensively studied [3–7]. It has been widely deployed in industry as demonstrated in [8] which shows that about 25% gates are buffers in recent IBM ASIC designs. On the other hand, interconnect resistivity may lead to the significant degradation on signal integrity. This issue severely aggravates with advancing technologies. As such, buffers need to be inserted also for meeting slew constraints [2].

In practice, slew constraint is significantly more prevalent than timing constraint [2]. Once nets are buffered for satisfying slew constraints, most of them will automatically satisfy timing constraints. In fact, it is documented in [8] that in IBM ASIC designs, for about 95% nets, buffering based on slew is sufficient to meet their timing constraints, while only about 5% nets need to be re-buffered for timing optimization.

This suggests a better way of using buffer insertion techniques in the physical synthesis flow [8]. Suppose that we are to buffer millions of nets. They are first buffered using slew driven buffering techniques. After performing timing analysis on the resulting nets, we find that about 5% nets violate timing constraints. Only these timing critical nets need to be ripped up and re-buffered by timing driven buffering techniques [2].

The main benefit from this new physical synthesis methodology is the huge gain in efficiency since slew buffering can be performed very efficiently. It is demonstrated in [2] that a slew driven buffering algorithm can run up to 88× faster than the timing driven buffering algorithm. For example, one can buffer 1000 industrial nets in only 6.2 s by the minimum cost slew buffering algorithm, while the minimum cost timing buffering algorithm needs 548.9 s. Note that minimum cost buffering is important since excessive buffers may cause many design issues such as high power consumption. To keep the overall design quality, it is crucial to minimize the usage of buffering resources [2].

This fast slew buffering algorithm proposed in [2] needs an important assumption, namely, the input slew to each buffer is assumed to be fixed at a conservative upper bound. With this input slew assumption, slew buffering can be efficiently performed under the dynamic programming framework. Certainly, improvement in buffer area is desired if this assumption is

[☆] A preliminary version of the paper appeared in [1].
[*] Corresponding author. Tel.: +1 906 487 2941; fax: +1 906 487 2949.
E-mail addresses: shiyan@mtu.edu (S. Hu), jianghu@ece.tamu.edu (J. Hu).

0026-2692/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2009.08.003

eliminated. As such, they [2] also propose a slew buffering algorithm without the fixed input slew assumption. The idea is to first discretize every possible input slew into slew bins and carry out the fixed input slew buffering algorithm with each bin. Since the solutions associated with a slew bin may be switched to other slew bins, numerous solutions can be generated. Experimental results in [2] show that although about 20% area saving can be obtained, the algorithm is not efficient: it is even slower than timing driven buffering in many cases. Thus, it is not very attractive since a major reason for using slew buffering is its high efficiency. In order to make the approach practical, it is crucial to design a fast slew buffering algorithm for handling the non-fixed input slew case.

This work proposes a new fast slew buffering algorithm without input slew assumptions. In contrast to [2] which builds slew buffering solutions from scratch, we perform optimizations to buffering solutions obtained with the fixed input slew assumption. For this purpose, a heuristic is proposed to improve buffer usage under the slew constraint and it runs very fast. Together with the fact that slew buffering with fixed slew assumption can be efficiently computed, the whole approach runs very fast.

Our experimental results demonstrate the effectiveness and the efficiency of the new algorithm. Our algorithm runs up to 49× faster than the timing buffering algorithm with about 10% buffer area saving. Compared to the slew buffering algorithm without input assumptions proposed in [2], up to 37× speedup is obtained. Thus, our work makes the general slew buffering technique practical. It is expected that the new algorithm would be widely used in practice due to its high efficiency in both runtime and buffer usage.

Note that there is another recent work in [9] which proposes a low-power buffering algorithm handling both timing constraint and slew constraint for timing critical nets. In contrast, the purpose of this paper is to address slew buffering on non-timing critical nets.

The rest of the paper is organized as follows: Section 2 formulates the slew buffering problem. Section 3 overviews the slew buffering algorithm proposed in [2]. Section 4 describes the new fast slew buffering algorithm without fixed input slew assumption. Section 5 presents the experimental results with analysis. A summary of work is given in Section 6.

2. Preliminaries

For completeness, we first introduce the slew problem as formulated in [2]. In the slew buffering problem, we are given a routing tree $T = (V, E)$. V consists of source vertex, sinks and internal vertices. Each sink has sink capacitance C_s. Each edge has lumped resistance R_e and lumped capacitance C_e. We are also given a buffer library B. Each type of buffer b has a cost W_b. At each internal vertex, some types of buffered can be inserted. A buffering solution is defined as a buffer assignment where buffers are inserted at some internal locations. The cost of a buffering solution γ is defined as $W(\gamma) = \sum_{b \in \gamma} W_b$ [2].

We are to compute a minimum cost buffering solution such that the slew constraint is satisfied. The signal slew is the measure of rising or falling time of switching. As in [2], $\frac{10}{90}$ slew is used which refers to the difference between the time signal waveform crosses the 90% point and the time signal waveform crosses the 10% point. The slew model can be described by the following generic example in [2]. Consider a path p from an upstream vertex u to a downstream vertex v. Assume that a buffer b is inserted at u and no buffer is inserted between u and v. Denote the output slew of b by $S_{b,out}(u)$ and the slew degradation along path p by $S_w(p)$. The slew $S(v)$ at v is computed as [10,2]

$$S(v) = \sqrt{S_{b,out}(u)^2 + S_w(p)^2}. \tag{1}$$

As the Elmore model for delay, the slew degradation along wire $S_w(p)$ can be computed by Bakoglu's metric [11] as

$$S_w(p) = \ln 9 \cdot D(p), \tag{2}$$

where $D(p)$ is the Elmore delay along p. The output slew of a buffer, such as $S_{b,out}(u)$, depends on the input slew at this buffer and its load capacitance. As in [2], the dependence is described by a lookup table.

In [2], a fast algorithm is proposed to handle a simplified slew buffering formulation where the input slew to each buffer is assumed to be fixed at a conservative upper bound. This assumption allows us to process large number of nets very efficiently and slew constraint is satisfied. With the assumption, the output slew of buffer b at vertex v is then given by [2]

$$S_{b,out}(v) = R_b \cdot C(v) + K_b, \tag{3}$$

where $C(v)$ is the downstream capacitance at v, R_b and K_b are empirical fitting parameters. As in [2], we call R_b the slew resistance and K_b the intrinsic slew of buffer b.

To illustrate the above concepts, we use the example in Fig. 1 where a neighboring pair of buffers b_1, b_2 are connected by a path $p = (v_j, v_k)$. The slew rate at v_k is [2]

$$S(v_k) = \sqrt{S_{b_1,out}(v_j)^2 + S_w(p)^2}, \tag{4}$$

where $S_w(p)$ refers to the slew degradation along the wire p, and $S_{b_1,out}(v_j)$ is obtained through a 2-D look-up table when handling non-fixed slew buffering, while it is computed by Eq. (3) when handling fixed input slew buffering.

The slew buffering problem is formulated in [2] as follows.

Slew constrained minimum cost buffer insertion problem: Given a routing tree $T = (V, E)$, possible buffer positions, and a buffer library B, compute a buffering solution γ such that the total cost $W(\gamma)$ is minimized and the slew constraint α is satisfied.

3. Overview of [2]'s minimum cost slew buffering algorithm assuming fixed input slew

The algorithm proposed in [2] works under the dynamic programming framework as timing buffering [12,3]. For completeness, we include the algorithm in [2] as follows.

In the algorithm, a set of candidate solutions are propagated from the sinks toward the source. Each buffering solution γ is characterized by (C, W, S), where C denotes the downstream capacitance at the current node, W denotes the cost of the solution and S is the cumulative slew degradation along wire. S is S_w defined in Eq. (2). The solution at a sink node has C as sink capacitance, $W = 0$ and $S = 0$. During solution propagation, we will perform "add wire", "add buffer" and "merge branch".

"*Add wire*": To propagate a solution γ_v from a node v to its parent node u, a solution γ_u is generated at u as follows. $C(\gamma_u) = C(\gamma_v) + C_e$, $W(\gamma_u) = W(\gamma_v)$ and $S(\gamma_u) = S(\gamma_v) + \ln 9 \cdot D_e$ where $D_e = R_e(C_e/2 + C(\gamma_v))$.

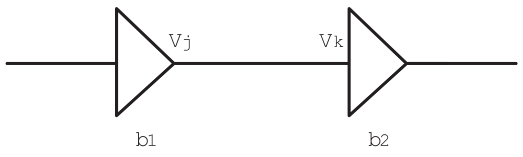

Fig. 1. Slew rate computation.