Fanout Optimization under a Submicron Transistor-Level Delay Model

P. Cocchini†, M. Pedram‡, G. Piccinini† and M. Zamboni†
†Politecnico di Torino, Torino, Italy
†cocchini@polito.it, piccinini@polito.it, zamboni@polito.it
‡Univ. of Southern California, Los Angeles, CA
‡massoud@zugros.usc.edu *

Abstract

In this paper we present a new fanout optimization algorithm which is particularly suitable for digital circuits designed with submicron CMOS technologies. Restricting the class of fanout trees to the so-called bipolar *LT*-trees, the topology of the optimal fanout tree is found by means of a dynamic programming algorithm. The buffer selection is in turn performed by using a continuous buffer sizing technique based on a very accurate delay model especially developed for submicron CMOS processes. The fanout trees can distribute a signal with arbitrary polarity from the root of the tree to a set of sinks with arbitrary required time, required minimum signal slope, polarity and capacitive load. These trees can be constructed to maximize the required time at the root or to minimize the total buffer area under a required time constraint at the root. The performance of the algorithm shows several improvements with respect to conventional fanout optimization methods. More precisely, the area and delay improvements are 28% and 7%, respectively, when the algorithm is applied to entire circuits.

1 Introduction

During logic synthesis, several design steps are performed to translate the initial logic description into a physical netlist suitable for the final manufacturing. One of these steps, *fanout optimization*, is usually required after the *technology mapping* step where typically, for a large number of nodes in the circuit, the output signal must be propagated to several destinations (or *sinks*). Theoretically, a fanout algorithm should be able to take advantage of the slack available at some outputs to increase the slack at the initially more critical outputs to achieve an equilibrium point where all outputs are equally critical. Conventional techniques commonly used for CMOS standard cells do not usually achieve this goal because of the discrete nature of the delay optimization they are based on. For example, the works reported in [1, 2, 4, 7, 8, 9] rely on a cell library with a finite number of available buffers. Furthermore they all use very simple delay models that severely limit their applicability especially when submicron processes are involved. On the other hand, the approach we present considerably improves these two aspects. Indeed, it is based on a *continuous* delay optimization technique whose main features are high accuracy and independence from the technology in use. The delay model is first applied to the creation of two numerical routines for the design of delay and area optimized CMOS tapered buffers. Then, a buffering algorithm uses them to create a fanout tree where the available slacks at the destinations are fully exploited to generate drivers whose delays are tailored to fit perfectly between the sink required times.

In Section 2 we give an overview of the delay model adopted in our work and introduce the routines used for the generation of the optimized buffers. In Section 3 we give some basic definitions and explain the buffering algorithm proposed for the solution of the fanout problem. Section 4 reports the results obtained testing the algorithm on different benchmark circuits. Concluding remarks are presented in Section 5.

*This research was funded in part by NSF PECASE award number MIP-9628999 and SRC under contract number 98-DJ-606.

2 Delay Optimization

2.1 Inverter Delay Model

Since the buffering process which we perform for the generation of a fanout tree only involves CMOS tapered buffers, we are interested in modeling the behavior of their basic component, that is a static CMOS inverter whose schematic is reported in Figure 1. The delay model that we use throughout the paper is the one presented in [3]. It is composed of a set of analytical equations which model the output response of a CMOS inverter taking into account the main second-order effects present in submicron processes. The input voltage and output voltage are modeled as signals with trapezoidal shape as shown in Figure 1. The *feed-through* effect between input and output is considered by means of a capacitance C_{FF}. We will not give here a detailed explanation of the delay equations as this is outside the scope of the present paper. For a more complete treatment, the reader is referred to [3]. In Figure 1 we also introduce some definitions used here and in the rest of the paper. With k_r and k_f we denote the slopes in [V/ns] of the rising and falling edges of a ramp shape voltage signal, respectively. Following this notation, k_{r_I} and k_{f_I} are the slopes of the input voltage V_I of the inverter, while k_{r_O} and k_{f_O} are the slopes of the output voltage V_O. Moreover, t_{pr} and t_{pf} are the propagation times of the rising and falling edges of V_O, respectively. They are

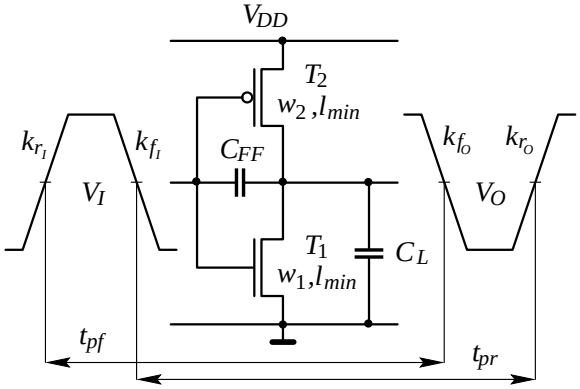

Figure 1: CMOS Inverter.

measured as the difference between the times where V_O and V_I are at 50% of their total swing. An inverter I is identified by the tuple $I = \{m, u\}$, where m is the ratio between the width w_1 of the pull-down transistor T_1 and the minimum width w_{min} allowed by the user, and u is the ratio between the widths of the pull-up and pull-down transistors of the inverter. With P we denote a set of process and layout parameters of the technology in use on which the delay model depends. In this context, the equations for the delay model can be represented as: $t_{pr} = f_1(P, k_{r_I}, k_{f_I}, C_L, m, u)$, $t_{pf} = f_2(P, k_{r_I}, k_{f_I}, C_L, m)$, $k_{r_O} = f_3(P, k_{r_I}, k_{f_I}, C_L, m, u)$, $k_{f_O} = f_4(P, k_{r_I}, k_{f_I}, C_L, m)$, where f_1, f_2, f_3, f_4 are non-linear functions of their arguments. To automatically perform the design of an inverter and therefore of a tapered buffer, these functions have been arranged in the routine `delay_INV`, written in C language, which can perform two different tasks:

Task 1 *Given P, k_{r_I}, k_{f_I}, C_L, m, u, calculate k_{r_O}, k_{f_O}, t_{pr}, t_{pf}.*

Task 2 *Given P, k_{r_I}, k_{f_I}, C_L, m, calculate u, k_{r_O}, k_{f_O}, t_{pr}, t_{pf} such that $t_{pr} = t_{pf} = t_p$.*

In Task 1, `delay_INV` simply computes functions f_1, f_2, f_3, f_4 for the given arguments. On the other hand, in Task 2, `delay_INV` first solves the non linear equation

$$f_1(P, k_{r_I}, k_{f_I}, C_L, m, u_x) = f_2(P, k_{r_I}, k_{f_I}, C_L, m) \quad (1)$$

for the width ratio u_x and then computes the remaining functions f_3 and f_4.

2.2 Buffer Design

A scheme of the buffers used for driving a large capacitive load is reported in Figure 2. As can be seen, the circuit is composed of a cascade of N inverters each one scaled up by a factor of M with respect to the previous one (the first inverter has always minimum size). A buffer B is then defined as a set of N inverters $B = \{I_1, I_2, \ldots, I_N\}$. We extend here the definition of delays and signal slopes for the voltages V_I and V_O given in the previous section. A methodology for the determination of the optimal parameters N and M of a buffer with minimum and symmetrical propagation delay ($t_{pr} = t_{pf} = t_p$) is given in [3]. Here, after an initial step that characterizes a cascade of inverters with different sizes for each process in use, speed optimized tapered buffers are designed which uniformly distribute the overall propagation delay t_p along the chain for any given capacitive load

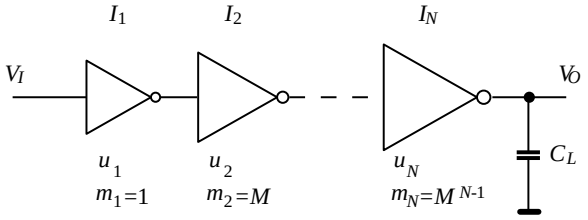

Figure 2: CMOS Tapered Buffer.

C_L. A limitation of this buffer optimization technique is that it considers only typical values for the input slopes k_{r_I} and k_{f_I}. Thus, to overcome this problem and consider arbitrary input slope values, the design of a minimum delay buffer is performed in this work by means of a new routine `min_delay_BUF` which is capable of performing the following task:

Task 3 *Given P, k_{r_I}, k_{f_I} and C_L, find a buffer B with minimum and symmetrical propagation delay $t_{pr} = t_{pf} = t_p$.*

Similarly, the design of a buffer with minimum area, subject to a time constraint in terms of maximum propagation delay, is accomplished by means of routine `min_area_BUF` that performs the following task:

Task 4 *Given P, k_{r_I}, k_{f_I}, C_L, $t_{p_{max}}$, $k_{r_{req}}$, $k_{f_{req}}$, find a buffer B with minimum area such that $t_{pr} = t_{pf} = t_p \leq t_{p_{max}}$, $k_{r_O} \geq k_{r_{req}}$ and $k_{f_O} \geq k_{f_{req}}$.*

Additional constraints on the minimum slope of the rising and falling edges of the output signal are also given in order not to worsen the delay of successive stages. Both routines `min_delay_BUF` and `min_area_BUF` are based on iterative calls to routine `delay_INV` which is used to compute the exact delay of each stage. Thus, the propagation delays t_{pr} and t_{pf} and the slopes k_{r_O} and k_{f_O} at the output of a buffer B can be put in the form: $t_{pr} = f_5(P, k_{r_I}, k_{f_I}, C_L, N, M, u_N)$, $t_{pf} = f_6(P, k_{r_I}, k_{f_I}, C_L, N, M)$, $k_{r_O} = f_7(P, k_{r_I}, k_{f_I}, C_L, N, M, u_N)$, $k_{f_O} = f_8(P, k_{r_I}, k_{f_I}, C_L, N, M)$, where f_5, f_6, f_7, f_8 are non linear functions, N is the the number of stages, M is the tapering factor, and u_N is the width ratio of the last inverter of buffer B. The values for the width ratio u of all the other stages are not specified as they remain fixed to default values. In the case of task 3, routine `min_delay_BUF` simply calculates the parameters N and M of the minimum delay buffer B according to the technique reported in [3], and then re-shapes its last stage solving the equation

$$f_5(P, k_{r_I}, k_{f_I}, C_L, N, M, u_N) = f_6(P, k_{r_I}, k_{f_I}, C_L, N, M)$$

for the variable u_N, in order to have a symmetrical output response. On the other hand, routine `min_delay_BUF` determines the minimum number of stages N_{min} and the corresponding parameter M of a tapered buffer whose propagation delays t_{pr} and t_{pf} are less then a given maximum value $t_{p_{max}}$. In particular, to find a buffer with minimum area and delay $t_{pf} \leq t_{p_{max}}$, `min_delay_BUF` first solves the non linear equation

$$t_{p_{max}} = f_6(P, k_{r_I}, k_{f_I}, C_L, N_{min}, M_{min})$$

for M_{min} such that $M_{min} \geq 1$, and then calculates the variable $u_{N_{min}}$, solving

$$t_{pf} = f_5(P, k_{r_I}, k_{f_I}, C_L, N_{min}, M_{min}, u_{N_{min}})$$

where $t_{pf} = f_6(P, k_{r_I}, k_{f_I}, C_L, N_{min}, M_{min})$, to have a symmetrical buffer output response. Finally, if the limits $k_{r_{req}}$ and $k_{f_{req}}$ on the output slopes are specified, functions f_7 and f_8 are computed to verify that the requirements of task 4 are met. Therefore if $k_{r_O} \leq k_{r_{req}}$ or $k_{f_O} \leq k_{f_{req}}$, the buffer with minimum area is designed solving the equations: $k_{r_{req}} = f_7(P, k_{r_I}, k_{f_I}, C_L, N_{min}, M_a, u_{N_{min}})$, $k_{f_{req}} = f_8(P, k_{r_I}, k_{f_I}, C_L, N_{min}, M_b)$ and taking $M_{min} = \max(M_a, M_b)$.

3 Fanout Optimization

Like other proposed fanout optimizations [2] [7] [8], our methodology relies on ordering sinks by non-decreasing required time. While restricting the set of all the possible fanout trees, this assumption allowed us to develop an efficient algorithm of polynomial complexity using dynamic programming. Apart from the far more accurate delay model, our optimization technique has other important advantages. First of all, there is not a buffer selection process where trees with same topology lead to different solutions because of the several combinations of distinct buffers available in a library. As a matter of fact, given a tree topology, the extent of the slacks between distinct leaves uniquely identifies the shape and size of the needed buffers. Secondly, the treatment of sinks with different polarities is intrinsically implemented in the fanout algorithm and does not increase its complexity. Finally, the adoption of a pre-processing step, which is presented in Section 3.7, can significantly reduce the number of distinct sinks to be driven so that the execution time of the algorithm is drastically shortened.

3.1 Definitions

We define S as the set of n destinations or sinks where a signal v, corresponding to the root of a tree, must be propagated. Each sink $s_i \in S$ has arbitrary polarity $p_{s_i} \in \{+, -\}$, capacitive load l_{s_i} and required time r_{s_i}. Furthermore, sinks $\{s_1, s_2, \ldots, s_n\}$ of S are ordered by increasing required time, that is, $\forall i \in [2, n-1]$, $r_{s_{i-1}} \leq r_{s_i} \leq r_{s_{i+1}}$. A group $G^p_{i,j} \subset S$ is then defined as the set of sinks of polarity p among the adjacent sinks $\{s_i, \ldots, s_j\} \subset S$, $l^p_{i,j}$ being the sum of the loads of its elements. Each group $G^p_{i,j}$ can be driven by a corresponding buffer B^p_i, whose input b^p_i has required time $r_{b^p_i}$ and load $l_{b^p_i}$ equal to the input capacitance of a minimum inverter, that is the one of its first stage. Finally, a fanout tree is defined as the set $T = \cup_i B^p_i$ of buffers B^p_i that form a tree where the leaves are groups and the union of all leaves equals S. Under these definitions, the fanout problem can be specified in two different ways depending on the cost function to be minimized.

Problem 1 (Max required time with Min area)
Build a tree T of buffers that distributes the signal v to the sinks S and 1) maximizes the required time r_v at its root, 2) minimizes the area of its implementation.

Problem 2 (Min area under required time constraint)
Build a fanout tree T that minimizes the area of its implementation such that the required time r_v at the root is $r_v \geq r_{v_{min}}$ where $r_{v_{min}}$ is a given minimum value.

Additional constraints to these problems are the specification of a minimum signal voltage slope at the sinks as well as the minimum slopes k_{r_v} and k_{f_v} of the signal to be propagated.

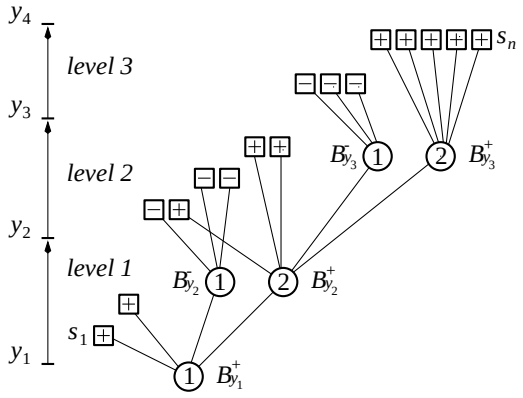

Figure 3: Example of tree topology with three distinct levels.

3.2 Tree Search Space

In order to reduce the complexity of the algorithm only a subset of all the possible trees must be considered. A scheme representing the topology of a fanout tree belonging to such a subset is reported in Figure 3. In this representation, sinks $S = \{s_1, s_2, \ldots, s_n\}$ are reported in order of increasing required time along the vertical axis, with the indication of their polarity, while buffers are drawn as small circles annotated with the number of stages they are composed of. A tree is divided into a set of z different levels identified by a (z+1)-tuple of integers $(y_1, \ldots, y_{z+1})$ such that: $y_1 = 1 < y_2 < \ldots < y_z < y_{z+1} = n+1$, with $1 \leq z \leq n$. Each level $i \in \{1, \ldots, z\}$ contains $y_{i+1} - y_i$ sinks, from s_{y_i} to $s_{y_{i+1}-1}$. Sinks with positive polarity form the group $G^+_{y_i, y_{i+1}-1}$ and are driven by a buffer $B^+_{y_i}$ whereas those with negative polarity form the group $G^-_{y_i, y_{i+1}-1}$ and are driven by a buffer $B^-_{y_i}$. In the case of Figure 3, $z = 3$ with a (z+1)-tuple (1, 3, 9, 17). Each buffer can accept a connection from one or two buffers belonging to the upper level $i+1$. Depending on the polarities of its sinks and the buffers of the upper level $i+1$, it follows that a level i can always have exactly one or two buffers driving its sinks. The class of trees that we have just defined is very similar to that of LT-Trees of type 1 introduced in [8]. While the trees belonging to such class have at most one buffer in the fanout of any buffer, in our case each buffer can drive 1 or 2 buffers along with any number of leaves. For this reason, we call our trees bipolar LT-Trees, or for short Bi-LT-Trees. Because of this property, it is apparent that each (z+1)-tuple identifies 2^z possible fanout trees. The number of possible (z+1)-tuples of integers corresponds to the number of distinct ways of choosing $z-1$ elements among $n-1$. Therefore, the total number of possible fanout trees is

$$\sum_{z=1}^{n-1} \binom{n-1}{z-1} 2^z = 2 \sum_{z=0}^{n-2} \binom{n-1}{z} 2^z = 2 \cdot 3^{n-1} - 2^n \quad (2)$$

Such search space is greater than both that of LT-Trees of type 1 (2^{n-2}), and LT-Trees of type 2 (2^{n-1}) [8]. In (2) we also assume that the first level can have two buffers which are driving sinks of different polarities. It is apparent that this situation is in contrast with the requirement of a one-rooted fanout tree like the one of Figure 3. Nevertheless, every occurrence of this kind can be uniquely resolved introducing one or two additional inverters in case $p_{b^+_1} \cdot p_{b^-_1} = +$ or $p_{b^+_1} \cdot p_{b^-_1} = -$, respectively, so that the equation still holds.

3.3 The Algorithm for Tree Selection

The selection of the best tree for the solution of Problems 1 and 2 is performed with the algorithm *tree_selection*, detailed in Figure 4. At the beginning, the process database P is loaded and sinks are ordered by non-decreasing required time. Then, the load $l^p_{i,j}$ of each possible group $G^p_{i,j} \subset S$ is pre-computed. The problem is now split in n sub-problems, identified by an index z, of sinks $(s_z, \cdots, s_n)$. A sub-problem z, then, is solved in $n - z + 1$ different ways, indicated by an index h, of which only the best one T_z is kept in a table, hence this is a dynamic programming approach. Each solution h corresponds to the insertion of one or two buffers B^+ and/or B^-, which respectively drive groups $G^+_{z,h}$ and $G^-_{z,h}$, and the upper level sub-tree T_{h+1}. Since the algorithm proceeds with z from n to 1, T_{h+1} has already been computed and is available. For each polarity $p \in \{+, -\}$,

```
algorithm tree_selection
load P, S, k_{r_req}, k_{f_req}, r_{v_min};
Sort S by increasing required time, S = {s_1, s_2, ..., s_n};
∀i ∈ [1,n], ∀j ∈ [i,n], ∀p ∈ {+,-}, l^p_{i,j} = Σ_{k=i}^{j} l_{s_k} δ_{p p_{s_k}},
where δ_{p p_{s_k}} is the Kronecker delta function;
for z = n to 1 {
  area(T_z) = +∞; r_{T_z} = -∞;
  for h = z to n {
    foreach polarity p ∈ {+,-} {
      load = l^p_{z,h} + l^p_{T_{h+1}};
      if (load > 0) {
        r_load = load required time;
        if (z > 1) then r_prev = r_{s_{z-1}};
        else {
          if (Problem = 1) then r_prev = r_load;
          else if (Problem = 2) then r_prev = r_{v_min};
        }
        B^p = min_area_BUF (P, k_r, k_f, load, r_load - r_prev, k_{r_req},
k_{f_req});
        if (B^p = ∅) then B^p_z = min_delay_BUF (P, k_r, k_f, load);
      } else B^p = ∅;
    }
    T = T_{h+1} ∪ B^+ ∪ B^-;
    r_T = min(r_{b+}, r_{b-});
    if (r_T > r_prev) {
      if (area(T) < area(T_z)) then T_z = T;
    } else {
      if (r_T > r_{T_z}) then T_z = T;
    }
  }
}
end tree_selection
```

Figure 4: The algorithm for the fanout tree selection.

the load of a buffer B^p is calculated as the sum of the pre-computed quantity $l^p_{z,h}$ and the load of same polarity $l^p_{T_{h+1}}$, offered by the sub-tree T_{h+1}. If such load is null, the corresponding buffer B^p is not inserted. Each buffer is designed calling the routine `min_area_BUF` whose arguments are ordered, and have the same meaning, as in the definition of task 4. Particularly, the slopes k_r and k_f of the input signal are chosen as typical values for a correct execution of the algorithm. As can be seen, the maximum allowed delay time $t_{max} = r_{load} - r_{prev}$ is equal to the difference of two terms: the required time r_{load} of the load driven by the buffer, and r_{prev} which is equal to the required time $r_{s_{z-1}}$ of the closest not yet buffered sink s_{z-1}. In this way, buffer B^p will have a required time equal or higher than $r_{s_{z-1}}$, thus not affecting the required time of subsequent sub-trees, and minimum area for its implementation. If t_{max} is too low and no buffer with such delay is possible, then B^p is designed by means of routine `min_delay_BUF`, which, given its arguments defined as for task 3, returns a minimum delay buffer. At this point, the h solution T of sub-problem z is formed by the union of buffers B^+, B^- and sub-tree T_{h+1}. If the required time r_T of T, defined as the minimum of the required times of B^+ and B^- (or the required time of one of them if the other is empty), is higher than r_{prev}, and its area is lower than the one of the best current solution T_z, then sub-tree T takes its place. On the other hand, if r_T is lower than r_{prev}, T is stored only if its required time is the highest. The same procedure applies to both Problems 1 and 2 until the last sub-problem $z = 1$, which corresponds to the overall fanout problem, has to be solved. As can be seen, in such a situation the required time t_{prev} takes different values. When Problem 1 is being solved, then $r_{prev} = r_{load}$ and buffers B^p are designed for minimum delay. On the other hand, for Problem 2, r_{prev} takes the value $r_{v_{min}}$, the given minimum required time of the root that can be exploited by the routine `min_area_BUF` to obtain a buffer with lower area. In this way, at the end of the process, tree T_1 stores the best solution for a given fanout problem.

3.4 Optimality for the Min Delay Problem

The optimality of the algorithm for the solution of Problem 1 is proved by the following theorem:

Theorem 1 (Optimality for Minimum Delay)
The `tree_selection` *algorithm produces an optimal fanout tree for Problem 1 over the class of all* Bi-*LT*-Trees, *assuming that routine* `min_delay_BUF` *produces optimal solutions to Task 3.*

Proof From the property of dynamic programming algorithms, the solution to Problem 1 is optimal exactly if such is true for the solution T_z of each sub-problem z. Therefore, for what pertains to the proof of the theorem, it is sufficient to prove the optimality of a single sub-tree T_z. The rest of the proof follows by induction on z. The solution of a sub-problem z, takes the generation of $n-z+1$ different sub-trees by means of routines `min_delay_BUF` and `min_area_BUF`. In each case, `min_area_BUF` introduces a buffer whose required time is always greater than the required time r_{prev} of the highest sink in the lower level. On the other hand, `min_delay_BUF` generates a speed optimized buffer whose delay is the smallest possible. The solution T_z is then chosen as the one with the highest required time r_T if every sub-solution has required time $r_T < r_{prev}$; otherwise the sub-tree with minimum area is taken. As a result, the solution T_z is optimal because it will offer to the next sub-problem $z - 1$, the smallest load to drive (the input capacitance of a buffer is always that of a minimum size inverter), with a required time such that the required time $r_{T_{z-1}}$ of the root of the subsequent sub-tree T_{z-1} can be the maximum possible. ■

3.5 An Example of Generated Tree

An example of a fanout tree generated by the algorithm *tree_selection* is reported in Figure 5 for a typical problem with 18 sinks and a 0.5μm CMOS process. Here, the required time of sinks and buffers is proportional to their position along the y-axis. As can be seen, there are three levels. Level 1 is composed of sinks s_1, s_2, s_3 and the inverter B^+_1, whereas level 2 is composed of sinks s_4, s_5 and the inverter B^-_4, and level 3 is composed of sinks s_6 through s_{18} and the two-stage buffers B^+_6 and B^-_6. It is interesting to note that the required time of both buffers B^+_6 and B^-_6 is greater

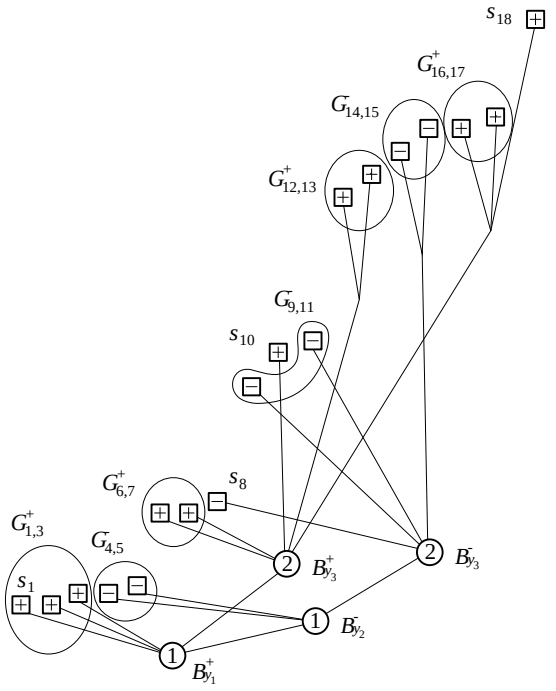

Figure 5: Fanout tree for a typical problem: $n = 18$, $z = 3$, $y_1 = 1$, $y_2 = 4$, $y_3 = 6$, $y_4 = 18$. Sinks can be merged into groups during the `merge_sinks` pre-processing step.

than that of any of the sinks belonging to the lower level 2. A thorough examination of the tree generation process indicates that both buffers have been designed to have minimum area through the routine `min_area_BUF`, and that the minimization process stopped because of the constraints on the minimum slopes $k_{r_{req}}$, $k_{f_{req}}$ of the output signal.

3.6 Complexity

The number of times we go through the most nested inner loop of the `tree_selection` algorithm is equal to $n(n+1)$. Therefore the complexity of the algorithm is $O(n^2)$, as we assume that both routines `min_delay_BUF` and `min_area_BUF` have complexity $O(1)$ and perform their respective tasks in constant time. When treating sinks of different polarities simultaneously, the algorithm proposed in [8] has complexity $O(d^2 \max(n,\ p) \max(np, \max(n,\ p)^{1.5}))$, while the one proposed in [7] has complexity $O(d^3\, n^2\, p^2)$. Here, d is the number of different buffers in the cell library, and n and p are the number of sinks of negative and positive polarity, respectively. As can be seen, our algorithm has smaller complexity due to the direct selection of buffers in the chosen trees.

3.7 Pre-Processing and Post-Processing

With our methodology sinks are treated independently of their load and there are no limits imposed on their size. This property suggests that sinks with equal or very close required time can be merged together to reduce the size of the problem with no adverse impact on the final result. An example of application of this technique, performed by the routine `merge_sinks`, is shown in Figure 5. Here, the number of distinct sinks n is now reduced to only 10, 7 of them corresponding to groups $G_{i,j}$ of sinks of the same polarity. Since this technique makes a great improvement in the computation time of the algorithm at no performance cost, it is always used during a pre-processing step to reduce the number of distinct sinks of a fanout problem. It has already been pointed out that during the execution of the algorithm `tree_selection`, the slopes k_r and k_f of the buffer input signal are chosen as typical values. This introduces some error, although small. After the algorithm has completed its execution and the topology of the best tree is available, the delay and slopes of all the buffers of the tree can be recomputed, yielding exact values, traversing the tree from root up.

4 Results and Verification

To provide experimental evidence of the efficiency of the proposed methodology, we have applied our algorithm to the fanout optimization of a set of benchmark circuits and compared its performance with those of the corresponding optimization algorithms available in SIS [6]. In the SIS environment, logic synthesis and minimum delay technology mapping steps have been performed for each circuit, using a custom 0.5μm CMOS process cell library calibrated in **DSMlib** (Deep Sub-Micron library) format. In this format, the delay of each pin of each cell is characterized by four subsets of 4 parameters each, modeling the propagation times t_{pr} and t_{pf}, and the transition times t_{tr} and t_{tf}. The transition times t_{tr} and t_{tf} are here defined as the difference between the times where the rising and falling edges of a signal are at 10% and 90% of their total swing, respectively. The pin-dependent delay model is as follows:

$$delay = (K_1 + K_2 \cdot load) \cdot transition\_time + K_3 \cdot load + K_4$$

where *delay* represents any of the terms t_{pr}, t_{pf}, t_{tr} and t_{tf} for the output pin, *load* denotes the capacitive load of the cell, and *transition_time* refers to t_{tr} or t_{tf} for the input pin as appropriate. The choice of such format has been dictated by the need for an accurate delay model which includes the effect of the slope of the voltage signals in the calculation of the standard cell timing. Notice that this delay model has only been used for the computation of the propagation and transition times during the timing analysis executed in the SIS environment[1]. The procedure used for the global optimization of a circuit is that presented in [5]. With this methodology every node is visited in topological order and when a fanout problem is encountered a fanout tree is introduced. In [8], this procedure is shown to be optimal with respect to delay minimization. The results of the global fanout optimization performed for minimum delay on the benchmark circuits are reported in Table 1. The **mapping** field reports the delay and area of the circuits after the execution of the technology mapping step for minimum delay. The second field reports the results of the best fanout optimization obtained from the spectrum of algorithms available in SIS (balanced trees, *LT*-trees, combinational merging, two-level trees, top-down traversal). The third field reports the results obtained by optimizing the circuits with the proposed **continuous** methodology. Finally, the last two columns report the performance comparison between the two approaches. Here, it must be pointed out that while the first approach selects, for each node, the best solution among those produced by each of the considered algorithms present in SIS, the **continuous** approach, in all cases, performs the optimization in the same way by means of the *tree_selection*

[1] In fact, here the concept of transition time is not contemplated at all.

	mapping		sis			continuous			% reduction	
circuit	delay	area	delay	area	cpu	delay	area	cpu	delay	area
9symml	5.78	72.2	4.90	167.8	87.2	4.71	105.4	2.7	4.0	37.2
C1355	6.22	255.6	6.18	479.3	109.8	6.00	335.7	1.7	3.0	30.0
C2670	8.22	343.0	6.88	656.5	190.1	6.53	487.1	4.0	5.4	25.8
C3540	16.02	523.7	13.12	1188.5	477.1	12.21	763.8	14.9	7.5	35.7
C5315	10.37	770.3	9.08	1551.7	505.2	8.01	1134.6	12.1	13.4	26.9
C6288	32.13	1504.1	29.65	3077.6	684.4	27.51	2111.6	30.9	7.8	31.4
C7552	23.33	993.8	11.85	2019.5	999.6	10.65	1437.4	31.8	11.3	28.8
alu2	9.36	165.4	8.20	380.8	120.0	7.48	243.2	3.8	9.6	36.1
alu4	12.09	306.9	10.58	703.4	233.2	9.86	456.7	7.9	7.3	35.1
apex6	7.57	319.3	5.42	604.1	184.9	5.24	485.7	3.5	3.4	19.6
apex7	5.02	113.4	3.67	214.1	109.0	3.58	169.0	1.1	2.5	21.0
comp	3.75	60.3	3.59	103.7	13.6	3.49	90.9	0.1	2.9	12.3
dalu	18.75	541.3	13.53	1214.0	461.2	12.74	819.5	17.6	6.2	32.5
k2	8.82	511.9	7.23	1101.4	262.8	6.81	804.0	6.5	6.2	27.0
misex3	8.44	241.8	6.79	610.3	313.3	5.96	358.5	12.1	13.9	41.3
rot	8.07	316.0	6.27	641.8	152.7	6.10	480.5	4.0	2.8	25.1
x2	2.49	34.7	2.25	60.8	21.6	2.19	51.8	0.3	2.7	14.7
x4	7.31	278.9	3.86	470.3	456.9	3.26	358.0	3.6	18.4	23.9
average	-	-	-	-	-	-	-	-	7.1	28.0

Table 1: Results for maximum speed fan-out optimization applied to entire circuits. Delay is the difference in nanoseconds between the required time at the first sink and the required time at the root, and cpu is the run-time in seconds on a SUN-Ultra 2. The tree area is given in $10^3 \mu m^2$.

algorithm. It must be pointed out that every buffer generated by the *tree_selection* algorithm was rounded up to the closest element in a set of 20 pre-designed buffers of different strengths which were available in the cell library. Here, instead of generating a new cell for each buffer, we have opted for a fixed number of buffers, available to the same extent to all fanout optimization algorithms, in order to make a more realistic and fair comparison. As can be seen, with the **continuous** approach every circuit is optimized in shorter time and the resulting implementation has lower delay and lower area. Particularly, the typical reduction is 7.1% in delay and 28% in area, while the computation time is typically one order of magnitude lower.

5 Conclusion and Future Work

In this paper we have presented a new methodology for the solution of the fanout problem based on a *continuous* delay optimization technique. An accurate transistor-level delay model is used to design delay and area optimized buffers that perfectly fit the slacks between the leaves of the fanout tree they set up, resulting in consistent area savings. Our approach is particularly effective for circuits developed with submicron CMOS processes where special care must be taken in the evaluation of delay times and signal slope effects. A polynomial time algorithm which uses dynamic programming for the selection of the best possible fanout tree has also been presented. The high accuracy of its delay model, the independence from the technology in use, the wide tree search space, and the fast run-time make the algorithm very convenient to be used in CAD tools for the automatic synthesis of digital circuits.

When deep submicron technologies are used, in several cases the resistance of the interconnections cannot be neglected and the performance of a fanout optimization technique can be further improved only with a post-placement approach where performance-driven fanout and routing optimization problems are solved simultaneously. We are currently investigating such an approach.

References

[1] T. Aoki, M. Murakata, T. Mitsuhashi, and N. Goto. Fanout-tree restructuring algorithm for post-placement timing optimization. In *ASP-DAC*, pages 417–422, August 1995.

[2] C. L. Berman, J. L. Carter, and K. F. Day. The fanout problem: From theory to practice. In C. L. Seitz, editor, *Proc. of the 1989 Decennial Caltech Conference*, pages 69–99. MIT press, March 1989.

[3] P. Cocchini, G. Piccinini, and M. Zamboni. A comprehensive submicrometer MOST delay model and its application to CMOS buffers. *IEEE J. Solid-State Circuits*, 32(8):1254–1262, August 1997.

[4] M. C. Golumbic. Combinatorial merging. *IEEE Transactions on Computers*, 25:1164–1167, November 1976.

[5] H. J. Hoover, M. M. Klawe, and N. J. Pippinger. Bouding fan-out in logical networks. *Journal of the Association for Computing Machinery*, 31(1):13–18, January 1984.

[6] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Proc. of ICCD*, pages 328–333, October 1992.

[7] K. J. Singh and A. Sangiovanni-Vincentelli. A heuristic algorithm for the fanout problem. In *Proceedings of the 27th DAC*, pages 357–360, June 1990.

[8] H. Touati. *Performance-oriented technology mapping.* PhD thesis, University of California, Berkeley, November 1990. Technical Report UCB/ERL M90/109.

[9] H. Vaishnav and M. Pedram. Routability-driven fanout optimization. In *Proceedings of the 30th DAC*, pages 230–235, June 1993.