Fanout Optimization under a Submicron Transistor-Level Delay Model

Pasquale Cocchini and Massoud Pedram

CENG 97-22

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4458

November 1997

Fanout Optimization under a Submicron Transistor-Level Delay Model

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, fanout trees are typically built with 73% and 20% lower area and delay, respectively, while the accuracy of calculated arrival times and signal slopes at the sinks have a typical agreement of 5% with SPICE simulations. These results are obtained for a library which consists of 20 buffers with different strengths.

1 Introduction

The objective of *fanout optimization* is to build circuits that distribute a signal to one or more destinations at a minimum cost. Given the individual times at which the signals are required at each destination, we can build a buffer tree that delivers the signals earlier and has the potential of further decreasing the delay through an entire circuit. 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, all the works in [3], [1], [6] and [8] 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 made possible by the delay model adopted from [2] 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. Although this methodology requires that each new cell

is inserted in the current library, we feel this does not constitute a limitation as all the major platforms nowadays available for the design of integrated circuits support tools for the automatic generation of cell layout (especially buffers and inverters). However, in case a library must contain a fixed number of cells and cannot be modified by the user, our technique can still be used, even though with less effectiveness, if each buffer generated by the optimization algorithm is rounded up to the closest corresponding element of that library.

In brief, the contribution of the present work can be summarized by the following points:

- An accurate technology independent submicron CMOS delay model is used for the computation of propagation times and output slopes.
- A novel *continuous* delay optimization technique based on speed and area optimized buffers is exploited.
- A restricted class of fanout trees, namely bipolar LT-trees is introduced. This class is larger than the class of LT-trees introduced in [8].
- A dynamic programming algorithm `tree_selection`, for the solution of the fanout problem is presented. Fanout trees can be found maximizing the required time of the root (while keeping the area at a minimum), or minimizing the area given a delay constraint.
- Over the the restricted class of bipolar LT-trees, the solution for fanout trees with maximum required time (keeping their area at a minimum) is found optimally in polynomial time. Notice that the general fanout optimization problem is NP-Complete [1].
- While treating sinks with different polarity simultaneously, the algorithm has lower complexity and better performances in terms of delay and area of the trees with respect to all other fanout optimization algorithms available in the SIS environment [5].
- Constraints on the minimum signal slope required at the sinks can be given so that the gates which are connected to the fanout tree can be driven correctly.

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 fanout problems extracted from common benchmark circuits. Concluding remarks are presented in Section 5.

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 [2]. 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

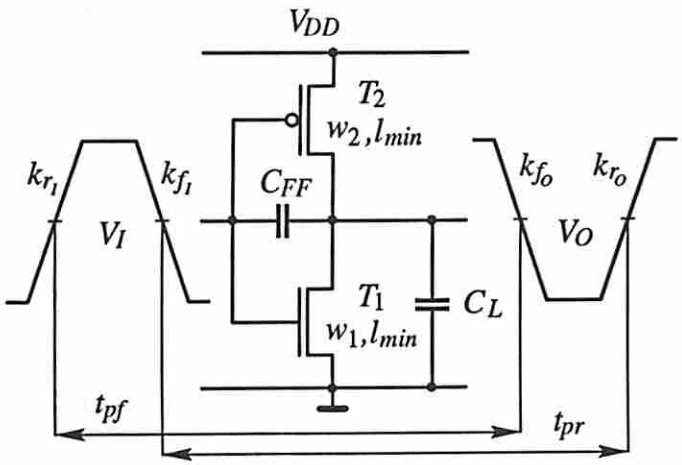

Figure 1: CMOS Inverter.

in Figure 1. The *feed-through* effect between input and output is considered by means of a capacitance C_{FF}. The equations depend on a small set of process parameters that can be conveniently extracted from SPICE model cards, therefore they are not tied up to any technology and do not require any calibration step. The main components of the delay mechanism are intrinsically included in the model so that the accuracy is only affected by approximations inherent to the equations. We will not give here a detailed explanation of the delay equations as this is outside the scope of the present paper. However, for a better comprehension of the work, a brief description of the mechanism with which the delay of an inverter is calculated is provided in Appendix A. For a more complete treatment, the reader is referred to [2].

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 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:

$$t_{pr} = f_1(P,\ k_{r_I},\ k_{f_I},\ C_L,\ m,\ u) \qquad t_{pf} = f_2(P,\ k_{r_I},\ k_{f_I},\ C_L,\ m)$$

$$k_{r_O} = f_3(P,\ k_{r_I},\ k_{f_I},\ C_L,\ m,\ u) \qquad k_{f_O} = f_4(P,\ k_{r_I},\ k_{f_I},\ C_L,\ m)$$

where f_1, f_2, f_3, f_4 are non-linear functions of their arguments (see [2] for the exact form of these functions). 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 2.1 *Given* P, k_{r_I}, k_{f_I}, C_L, m, u, *calculate* k_{r_O}, k_{f_O}, t_{pr}, t_{pf}.

Task 2.2 *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 2.1, `delay_INV` simply computes functions f_1, f_2, f_3, f_4 for the given arguments. On the other hand, in Task 2.2, `delay_INV` first solves the non linear equation

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

for the width ratio u_x and then computes the remaining functions f_3 and f_4. Here, it must be noted that (1) is solved with very few iterations of functions f_1 and f_2 as the delay model has the ability of providing an initial value for u very close to u_x.

The routine `delay_INV` has been applied to the calculation of the delays t_{pr} and t_{pf} for a minimum size inverter designed with a 0.7μm CMOS technology for a wide range of input voltage slopes and capacitive loads. While the typical agreement with SPICE simulations was 3% in the case of task 2.1 the routine was over 1000 times faster in terms of CPU time.

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 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 [2]. 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 C_L. For the convenience of the reader, a short description of this methodology is provided in Appendix B.

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 improves the technique of [2] and that is capable of performing the following task:

Task 2.3 *Given P, k_{r_I}, k_{f_I} and C_L, find a buffer B with minimum and symmetrical propagation delay $t_{pr} = t_{pf} = t_p$.*

As will be explained in Section 3, such a routine is usually executed when one or more sinks in a fanout tree have to be driven introducing a delay optimized buffer. In the situation where this delay could propagate to the root of the tree, it is important to assign a buffer with the minimum possible delay $t_{p_{min}}$ so that the required time at the source of the tree is decreased the least. Nevertheless, in many cases the slack between different buffered sinks can be of an extent such that the propagation delay t_p can be relaxed assuming a higher value $t_p > t_{p_{min}}$, so that a considerable amount of area can be saved. This situation, which recurs in most of the fanout problems commonly encountered during the automatic synthesis of digital circuits, is the key factor of the *continuous* delay optimization we propose, and can be exploited by means of routine `min_area_BUF` that performs the following task:

Task 2.4 *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 $k_{f_O} \geq k_{f_{req}}$.*

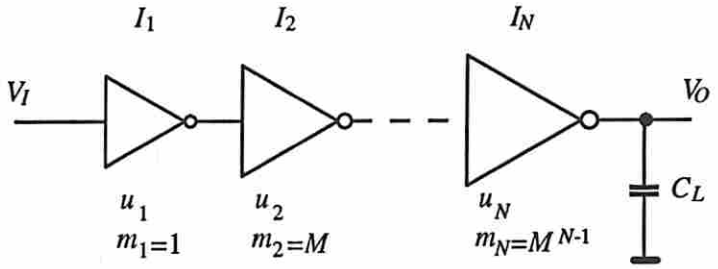

Figure 2: CMOS Tapered Buffer.

As can be seen, `min_area_BUF` is capable of designing a buffer with minimum area given a time constraint in terms of maximum propagation delay. 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

$$t_{pr} = f_5(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N,\ M,\ u_N) \qquad t_{pf} = f_6(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N,\ M)$$

$$k_{r_O} = f_7(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N,\ M,\ u_N) \qquad k_{f_O} = f_8(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N,\ M)$$

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 2.3, routine `min_delay_BUF` simply calculates the parameters N and M of the minimum delay buffer B according to the technique given in [2], and then reshapes its last stage solving the equation

$$f_5(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N,\ M,\ u_N) = f_6(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N,\ M) \tag{2}$$

for the 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

$$t_{p_{max}} = f_6(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N_{min},\ M_{min}) \tag{3}$$

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

$$t_{pf} = f_5(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N_{min},\ M_{min},\ u_{N_{min}}) \tag{4}$$

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 the output slopes are specified, functions f_7 and f_8 are computed to verify that the requirements of task 2.4 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

$$k_{r_{req}} = f_7(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N_{min},\ M_a,\ u_{N_{min}}) \tag{5}$$

$$k_{f_{req}} = f_8(P,\ k_{r_I},\ k_{f_I},\ C_L,\ N_{min},\ M_b) \tag{6}$$

and taking $M_{min} = \max(M_a,\ M_b)$.

Regarding the complexity of these routines, it must be pointed out that they are always capable of finding the solution to the corresponding set of equations after a small and limited number of iterations. Particularly, this property is achieved because in all cases, consistent initial values are provided by the buffer optimization technique of [2].

3 Fanout Optimization

Like other proposed fanout optimizations [1] [6] [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 3.5, can significantly reduce the number of distinct sinks to be driven so that the execution time of the algorithm is drastically shortened. In order to ease the task of describing the proposed methodology, in the following we give some definitions and formulate the fanout problems we consider in our work.

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 3.1 (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 3.2 (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.*

Notice that in Problem 3.1 we first optimize for the maximum required time, and then minimize the area at no cost for delay. In contrast, in Problem 3.2, given a minimum required time, we minimize the area subject to that constraint.

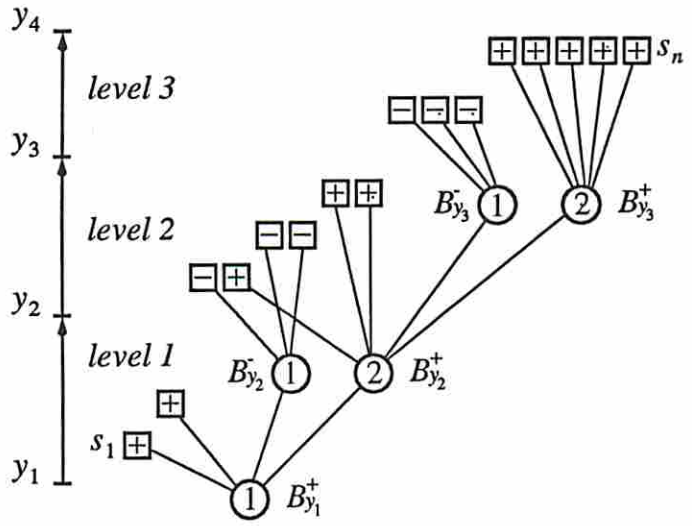

Figure 3: Example of tree topology with three distinct levels.

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.

3.2 Tree Search Space

In order to reduce the complexity of the algorithm only a subset of all the possible trees is considered. This subset is small enough to permit a fast generation of solutions and large enough to satisfactorily solve a large spectrum of fanout problems. 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 shortly 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

$$\sum_{z=1}^{n-1} \binom{n-1}{z-1} 2^z = 2 \sum_{z=0}^{n-2} \binom{n-1}{z} 2^z = 2 \cdot 3^{n-1} - 2^n \tag{7}$$

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 (7) 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 3.1 and 3.2 is performed with by 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 $p \in \{+,-\}$, 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 2.4. 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 2.3, 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 t_{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 t_{prev}, T is stored only if its required time is the highest.

The same procedure applies to both Problems 3.1 and 3.2 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 3.1 is being solved, then $r_{prev} = r_{load}$ and buffers B^p are designed for minimum delay. On the other hand, for Problem 3.2, r_{prev} takes the value $r_{v_{min}}$, the

```
algorithm tree_selection
load P, S, k_{r_req}, k_{f_req}, r_{v_min};
Sort S by increasing required time resulting in S = {s_1, s_2, ..., s_n};
∀i ∈ [1,n], ∀j ∈ [i,n], ∀p ∈ {+,−}, compute l^p_{i,j} = Σ_{k=1}^{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 {
  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 = 3.1) then r_prev = r_load;
          else if (Problem = 3.2) 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.

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.

Theorem 3.1 *The* `tree_selection` *algorithm produces an optimal fanout tree for Problem 3.1 over the class of all* Bi-*LT*-Trees, *assuming that routine* `min_delay_BUF` *produces optimal solutions to task 2.3.*

Proof From the property of dynamic programming algorithms, the solution to Problem 3.1 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

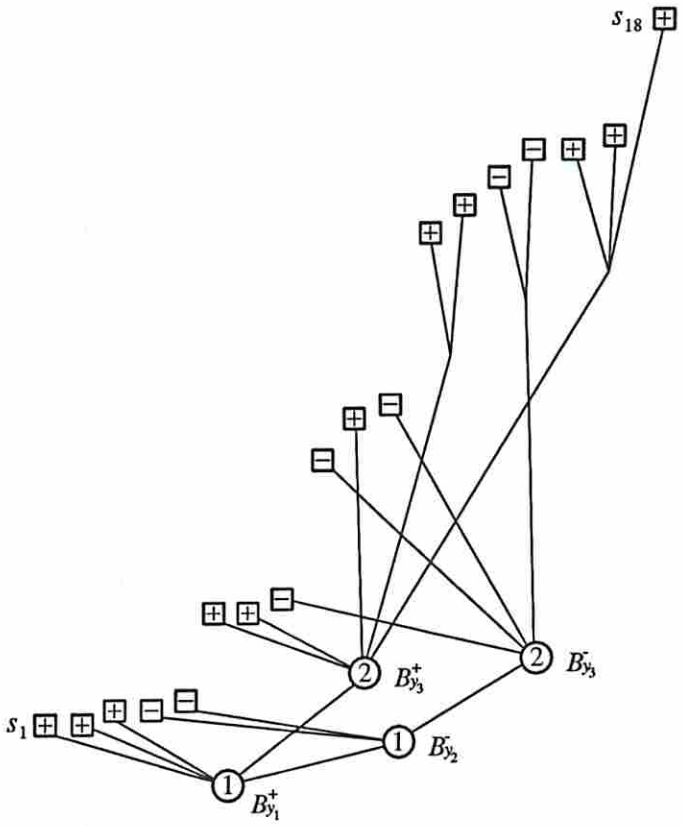

Figure 5: Fanout tree for a typical problem: $n = 18$, $z = 3$, $y_1 = 1$, $y_2 = 4$, $y_3 = 6$, $y_4 = 18$.

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. ■

Unlike the case of Problem 3.1, the `tree_selection` algorithm does not produce the optimal solution to Problem 3.2. Nevertheless, this shortcoming can be easily remedied by adopting a binning technique (similarly to the one used in [8]), modifying the algorithm at the cost of increased complexity. More precisely, for each tuple $(z,\ h)$ such that $z > 1$, the minimum required time r_{prev} given as argument to the routine `min_area_BUF` has to assume a discrete set of τ different possible values. Therefore, each sub-problem z results in a richer spectrum of $\tau(n - z + 1)$ sub-solutions T, each one with different area and required time, available for the generation of the fanout tree. In this way, neglecting the error introduced by the discretization of r_{prev}, the optimality for the solution to Problem 3.2 is achieved with complexity $O(\tau\, n^2)$. In practice, since we have found that this technique, even though it produces the optimal solution for Problem 3.2, does not provide substantial area improvements compared to the basic algorithm, we have chosen to exclude it from the *tree_selection* algorithm in order to retain a low complexity.

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 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. This situation, which is very recurrent in almost all of the standard-cell based fanout problems, is fully exploited by our buffering mechanism, leading to the generation of a fanout tree with the lowest area cost.

3.4 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 [6] 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.

3.5 Pre-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 to the test case of Figure 5, by means of the routine `merge_sinks`, is shown in Figure 6. Here, the number of distinct sinks n is now reduced to 10, 7 of them corresponding to groups G_i^j of sinks of the same polarity. As can be seen, the result in terms of speed and area of the fanout tree is the same as Figure 5. However, in this case the user CPU time needed by the computation is significantly lower. 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.

3.6 Post-Processing

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. Particularly, the output slopes k_{r_O}, k_{f_O} of the level 1 buffers are first calculated with the given input values k_{r_I} and k_{f_I} of Problems 3.1 and 3.2, and then reused as input values for the buffers of level 2. Iterating this process for the rest of the levels, the timing of the signal v distributed along the tree can then be accurately recomputed for all the intermediate nodes and destinations.

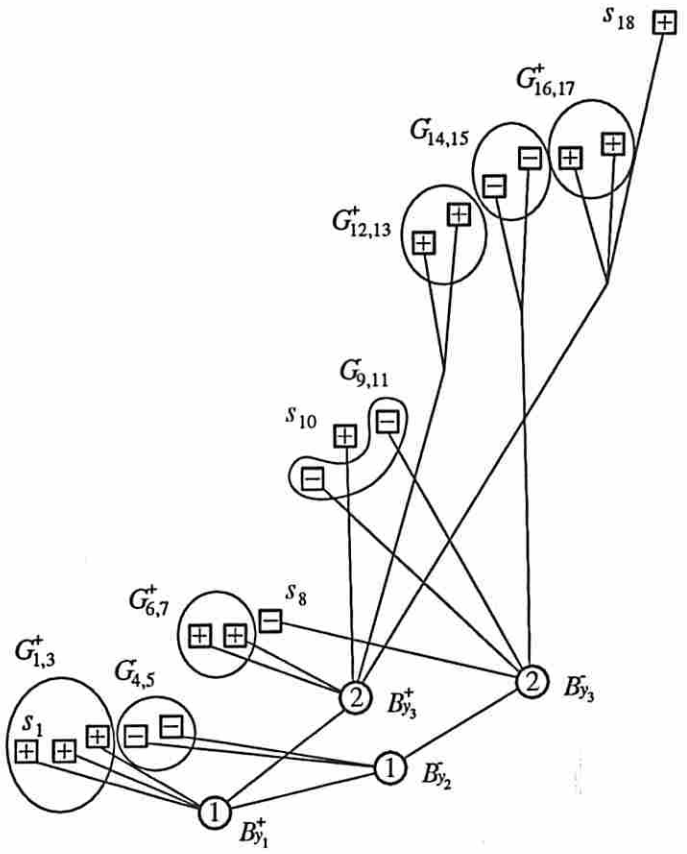

Figure 6: Fanout tree for a typical problem with the `merge_sinks` pre-processing step.

4 Results and Verification

To provide experimental evidence of the efficiency of the proposed fanout optimization, we have applied our algorithm to the solution of several fanout problems extracted from a set of six ISCAS benchmark circuits. For each circuit, logic synthesis and minimum area technology mapping steps have been performed in the SIS environment, using a custom 0.5μm CMOS process cell library calibrated in **DSMlib** (Deep Submicron 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 transistion 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.

From each circuit, four fanout problems of different complexity have been extracted and written in blif format for testing the algorithm and comparing its performance with those of SIS [5]. SIS provides

Problem		SIS				Our Algorithm				Comparison	
circuit	sinks	g	delay	area	cpu	g	delay	area	cpu	delay	area
C432-1	16	16	0.23	11286	17.8	2	0.23	1545	1.16	0.0	86.3
C432-2	6	6	0.21	3085	3.3	2	0.21	1474	0.06	0.0	52.2
C432-3	16	9	0.31	8588	16.1	4	0.31	3020	0.68	0.0	64.8
C432-4	10	11	0.18	6977	7.0	5	0.16	2948	0.52	11.1	57.7
C1355-1	8	10	0.44	5896	3.8	5	0.44	2948	0.21	0.0	50.0
C1355-2	9	8	0.56	5896	4.2	3	0.46	3616	0.12	17.9	38.7
C1355-3	9	7	0.50	6152	4.6	2	0.41	1545	0.03	18.0	74.9
C1355-4	13	7	0.53	4678	11.1	1	0.38	877	0.03	28.3	81.3
C3540-1	12	11	0.39	8314	7.1	3	0.39	2213	0.30	0.0	73.4
C3540-2	35	18	0.24	14581	80.9	2	0.24	1617	1.74	0.0	88.9
C3540-3	21	12	0.33	9257	23.2	3	0.33	3020	0.47	0.0	67.4
C3540-4	72	42	0.19	32077	188.2	4	0.17	6289	12.42	10.5	80.4
C5315-1	49	25	0.64	16706	95.3	5	0.49	4463	3.85	23.4	73.3
C5315-2	12	15	0.36	11129	9.3	3	0.27	2142	0.39	25.0	80.8
C5315-3	21	15	0.28	10849	16.6	6	0.23	4428	0.70	17.9	59.2
C5315-4	50	25	0.35	18841	123.3	7	0.35	6248	5.64	0.0	66.8
C6288-1	16	14	0.16	11143	17.8	2	0.16	1545	1.48	0.0	86.1
C6288-2	21	19	0.16	15173	32.2	2	0.16	1545	3.14	0.0	89.8
C6288-3	60	46	0.24	37567	169.6	7	0.24	5652	19.51	0.0	85.0
C6288-4	50	26	0.16	20610	126.0	2	0.16	1617	4.72	0.0	92.2
C7552-1	283	43	0.54	40916	3000.0	11	0.41	10934	45.82	24.1	73.3
C7552-2	12	10	0.26	7645	7.8	3	0.26	2142	0.71	0.0	72.0
C7552-3	16	11	0.23	8176	11.4	3	0.20	2142	0.52	13.0	73.8
C7552-4	23	16	0.18	12353	27.4	3	0.15	2142	2.27	16.7	82.7
average	-	-	-	-	-	-	-	-	-	8.6	73.0

Table 1: Results for delay optimized fanout trees. For each tree, g represents the number of gates (buffers and inverters), 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-Sparc 20. The tree area is given in μm^2.

a spectrum of different fanout optimization algorithms, each one based on a different approach: balanced trees, LT-trees, combinational merging, two-level trees, top-down traversal. Particularly, while our algorithm has been selected to minimize firstly delay and secondly area (Problem 3.1) of the constructed trees, all the other algorithms have been designed to minimize the delay and are then followed by an additional step of area recovery.

For each fanout problem, a program implementing the algorithm *tree_selection* found an optimal tree consisting of continuously sized buffers needed to distribute the input signal to the corresponding destinations. Next, an output netlist in blif format was generated. In this netlist, every buffer of the tree 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.

The performance of the resulting circuits have been then compared in the SIS environment with the

best result among those achieved by all other SIS fanout optimization techniques (i.e. for each fanout problem, all of the SIS fanout optimization algorithms are run and the best result obtained by any of them is reported in the Table). Here, it must be pointed out that all of the fanout optimization algorithms in SIS have been modified to take the **DMSlib** format and that have been followed by the area recovery options (-AFG options). The results are reported in Table 4.

Each item in the table reports the complexity of the fanout problem in terms of the number n of sinks. A solution is then characterized by the number g of gates (buffers and inverters) the tree is composed of, the delay of the tree in nanoseconds calculated as the difference between the required time r_{s_1} at the first sink (sinks are sorted in order of non-decreasing required times) and the required time r_v at the root, the area of the implementation in μm^2, and the user CPU time in seconds needed for the computation. Finally, the percentage improvements in delay and area achieved by our proposed algorithm with respect to the various SIS algorithms are reported. As can be seen, due to the adoption of routines `min_delay_BUF` and `min_area_BUF`, in all cases the `tree_selection` algorithm generates fanout trees with a consistently lower area and lower or equal delay. In particular, the average improvements in delay and area are 20% and 73%, respectively. Furthermore, because of its low complexity, in almost all cases the computation time of our algorithm is much lower than the corresponding time needed by the other algorithms. Finally, in order to verify the accuracy of the adopted delay model, each fanout tree has been simulated with the SPICE program. The average error on the calculation of signal delay and slopes at the sinks was 5%.

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.

Having tested our algorithm against other different techniques for a significant number of typical fanout problems, we will next apply our algorithm on entire circuits.

References

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

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

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

[4] G. Massobrio and P. Antognetti. *Semiconductor Modeling with SPICE.* McGrow-Hill, 1993.

[5] 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 R. Werner, editor, *VLSI in Computers & Processors: Proc. of the 1992 IEEE International Conference on Computer Design*, pages 328–333. IEEE Computer Society Press, October 1992.

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

[7] S. M. Sze. *Physics of Semiconductor Devices.* Wiley, 1981.

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

[9] Yannis P. Tsividis. *Operation and Modeling of the MOS transistor.* McGrow-Hill, 1987.

Appendix

A Delay Model

We refer here to the calculation of the propagation time t_{pf} of the inverter depicted in Figure 1, although the results can be applied to the evaluation of any other delay time. For a falling output transition, C_{FF}, the feed-forward capacitance, is equal to the overlap capacitance of the NMOS transistor T_1 plus the capacitance introduced by the PMOS transistor T_2. At $t = 0$, C_L is charged at V_{DD} and the input voltage is zero. For $t > 0$ the input voltage V_I increases as $V_I = K_I\, t$ until it reaches V_{DD} and then remains constant, while V_O, after an initial increase, decreases to reach a switching voltage $V_S = V_{DD}/2$. During the transition of V_O from V_{DD} to V_S, depending on the slope K_I of the input voltage, five different regions of operation of the NMOS transistor can be distinguished as shown in Fig. 7.

In region 0, for $0 < t < t_0$, the transistor is off and $\Delta_0 = t_0$ is the time that V_I needs to switch the transistor on while C_L is being charged by the current I_{FF} due to the high slew rate of the input voltage, so that V_O increases.

In region 1, for $t_0 < t < t_1$, T_1 is in saturation while the input voltage V_I is still increasing reaching for V_{DD}. If at the time $t_{DD} = V_{DD}/K_I$, the voltage V_I reaches V_{DD} when the transistor is still in saturation, then it enters region 2, otherwise if the transistor goes in linearity while V_I is still less than V_{DD}, region 3 is entered. In both cases $\Delta_1 = t_1 - t_0$ is the time needed for the transition.

In region 2, for $t_1 < t < t_2$, T_1 is in saturation and V_I is constant and equal to V_{DD}. After the time $\Delta_2 = t_2 - t_1$, with the decrease of V_O, the transistor leaves its own drain current saturation region to enter region 4.

For $t_1 < t < t_3$, T_1 works in region 3, where the transistor is in linearity and the input voltage has not yet reached the supply voltage. In this situation, if V_O reaches V_S during the time $\Delta_3 = t_3 - t_1$, then the overall propagation time t_{pf} is equal to $t_{pf} = t_3 - t_{DD}/2 = \Delta_0 + \Delta_1 + \Delta_3 - t_{DD}/2$. On the contrary, if V_I reaches V_{DD} while the output voltage is still less than V_S, then region 4 is entered.

In region 4, T_1 is in linearity and V_I is stuck at V_{DD}. The time needed by V_O to reach V_S is either $\Delta_4 = t_4 - t_2$ or $\Delta_4 = t_4 - t_3$ and the overall propagation time becomes $t_{pf} = t_4 - t_{DD}/2 = \Delta_0 + \Delta_1 + \Delta_2 + \Delta_4 - t_{DD}/2$ or $t_{pf} = t_4 - t_{DD}/2 = \Delta_0 + \Delta_1 + \Delta_3 + \Delta_4 - t_{DD}/2$, respectively.

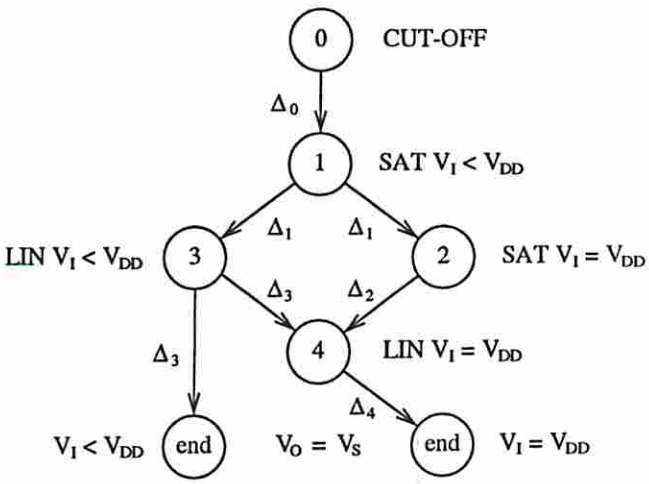

Figure 7: MOST regions of operation.

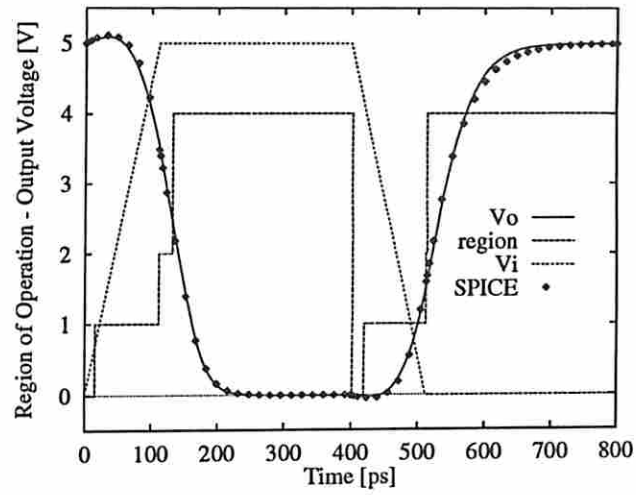

Figure 8: Output response of a minimum inverter.

The output response of a minimum inverter, loaded by another minimum inverter and driven by a trapezoidal shape input voltage, for a 0.7μm CMOS reference technology is reported in Figure 8 along with the result of a corresponding SPICE simulation. Here, a dashed line indicates the way the different regions of operation are crossed during the whole output transition. For a wide range of input voltage slopes and capacitive loads, the average error on the calculation of the propagation time is 3%.

B Buffer Optimization

Like other works based on tapered buffers, the optimization methodology we use relies on the assumption that the overall propagation delay of a speed optimized buffer is uniformly distributed along the structure. The scheme of such buffers, that we have already introduced in Section 2.2, is shown in Figure 2. We refer here to the definitions of delays and slopes given in the previous sections. For a more detailed treatment, the reader is referred to [2].

A thorough analysis of the circuit shows that under the above mentioned assumption, every stage $i \in \{1, \ldots, N\}$ has exactly the same behavior in terms of propagation delay t_{p_i} ($t_{p_i} = t_{pr_i} = t_{pf_i}$) and output slopes $k_{r_{O_i}}$ and $k_{f_{O_i}}$, provided that the buffer input signal has rising and falling slopes equal to those of the output, that is $k_{r_{O_N}} = k_{r_{I_1}}$ and $k_{f_{O_N}} = k_{f_{I_1}}$. Therefore, to predict the timing of the whole circuit, it is sufficient to model only one stage. To this purpose, the inverter delay model introduced in Section 2 is implemented in a one-step characterization algorithm, which gives all the needed relations between the tapering factor M, the width ratio u_i, and the output delays and slopes of the stage. This is done by means of four different fitted equations

$$t_{p_i} = a_1 + a_2\, M \qquad u_i = a_3 + a_4 \ln(M)$$

$$k_{r_{O_i}} = (a_5 + a_6\, M)^{-1} \qquad k_{f_{O_i}} = (a_7 + a_8\, M)^{-1}$$

where parameters $a_1, \cdots, a_8$ are the results of the characterization. In this way, a fast and accurate model of the basic stage is available for the optimization of the buffer delay.

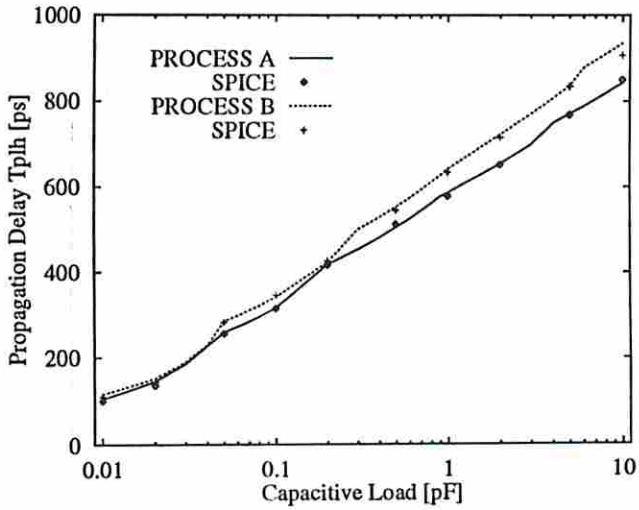

Figure 9: Delay optimized CMOS tapered buffers designed for two different 0.7μm technologies.

The overall propagation time t_p of the buffer is N times that of a single stage t_{p_i}. Thus, for uniformity of stage delays, we must have

$$t_p = N t_{p_i} = \frac{\ln(C_L/C_1)}{\ln(M)}(a_1 + a_2\, M)$$

where C_L and C_1 are the load capacitance and the stage input capacitance, respectively. By imposing the derivative of t_p with respect to M to be equal to zero, it is possible to find an expression for the optimized value of M which corresponds to the minimum overall propagation delay of the chain.

$$M_{opt} = \exp\left(1 + \frac{a_1}{a_2\, M_{opt}}\right) \tag{8}$$

Equation (8) can be easily solved by successive iterations considering that M_{opt} must be greater than e. The number of stages to be implemented in the buffer is then

$$N = \left\lfloor K_{as} \frac{\ln(C_L/C_1)}{\ln(M_{opt})} + 0.5 \right\rfloor$$

The coefficient K_{as}, which is usually taken to be close to unity, is introduced to decrease the number of stages in order to reduce the total amount of area used. By adjusting K_{as}, a good compromise between area and speed can be achieved.

After the determination of N, the optimum tapering factor must be recalculated, that is

$$M_{opt} = \left(\frac{C_L}{C_1}\right)^{\frac{1}{N}}$$

At this point, the width ratio u_i of each stage which provides a symmetrical output response of the buffer, and the overall propagation time t_p can be found by substituting M_{opt} in the equations previously introduced, so that

$$u_i = a_3 + a_4 \ln(M_{opt})$$

and

$$t_p = N(a_1 + a_2 M_{opt})$$

A design example carried out for two different 0.7μm CMOS processes, namely process A and B, is reported in Figure 9. In the same figure, the results of several SPICE simulation are also reported to test the accuracy of the calculated delay. As can be seen, for both processes A and B, the designed buffers show high precision for a wide range of load capacitances. Particularly, with C_L from 50fF to 10pF the agreement with SPICE simulations is better than 3%, while with smaller values of C_L the accuracy is about 6%.