# Layout-driven Timing Optimization by Generalized DeMorgan Transform


Supratik Chakraborty
IIT Bombay
India
supratik@cse.iitb.ernet.in

Rajeev Murgai
Fujitsu Laboratories of America, Inc
Sunnyvale, CA, USA
murgai@fla.fujitsu.com


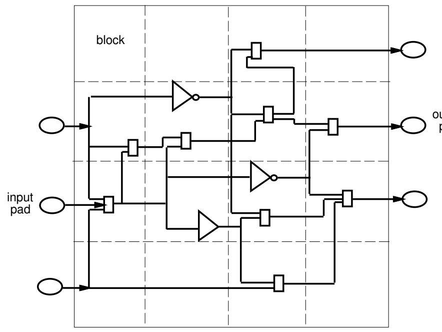


Figure 1: Block layout of a design


## Abstract

In this paper, we propose a new layout-driven, timing-oriented logic optimization technique called Generalized DeMorgan (GDM) transform. It is more general than the popular layout-oriented transforms such as gate resizing, net buffering, and more powerful than the simple DeMorgan transform. In fact, the GDM transform tightly integrates three transforms in one: gate resizing, generalized DeMorgan's laws, and net buffering, and thus is fairly powerful. It is layout-friendly in that it does not change the routing patterns and the placement positions of cells (except for some buffer insertions/deletions), and hence is useful in late stages of the design flow.

We propose a comprehensive GDM algorithm which a) determines the best GDM replacement of a gate along with the best buffering configurations of the nets incident on it, and b) embeds this into a global scheme for optimizing large designs. We have implemented it in our layout-driven, industrial-strength logic optimization framework [6], and have successfully applied it on large industrial designs. We show that typically one invocation of the GDM transform reduces the circuit delay more than net buffering alone, gate resizing alone, or a combination of net buffering followed by gate resizing. Experiments also show that the GDM transform, applied in concert with net buffering and gate resizing, yields the best possible delay results of all combinations of transformations in our repository.


# 1 Introduction

Timing convergence is an important problem in digital design, especially for emerging deep sub-micron technologies. To address this problem, logic synthesis and technology mapping tools must have access to layout information such cell as placements, wire lengths and wire delays, implying a tight coupling of physical and logical design flows. It is also becoming routine to apply simple transformations such as gate resizing, net buffering, gate replication etc. at design stages close to the layout process. These transformations, being simple, incremental and local in nature, do not change the layout significantly and are suitable for application just prior to detailed place-and-route. Towards this end, we have developed a tool for layout-driven logic optimization of high-speed digital circuits after global place-and-route but before detailed place-and-route [6]. In this paper, we describe a powerful technique, *Generalized DeMorgan* transform, implemented in this tool.

The input to our optimization tool is a circuit optimized by designers or logic synthesis tools and mapped to cells from a given cell library. It is assumed that a mincut-based algorithm has been used to partition the chip area into blocks using horizontal and vertical cuts, and each cell is assigned to some block (which may contain multiple cells). Further, we assume that a global routing tool has been used to determine the topology and global route of each net by generating net segments between blocks (Figure 1). We argue that this is a good stage in the design flow for applying optimization techniques such as net buffering, gate-resizing, etc. After cells have been roughly placed and nets globally routed, wire loads and delays are known more accurately, and so is the delay at each pin. However, since the final layout has not been frozen, any optimization obtained through careful addition/deletion/replacement of cells can be incorporated into the final layout by refined placement and detailed routing.

Earlier, we had implemented well-known transformations such as gate resizing and net buffering in our tool. In this paper, we propose a new technique called Generalized DeMorgan (GDM) transform. This is more general than gate resizing and net buffering in that it changes the logic functionality of gates in the design. It is also more powerful than the simple DeMorgan transform, which replaces a gate with its dual (such as an AND with an OR) and inserts inverters at the inputs and outputs. As we will see later, GDM transform integrates gate resizing, DeMorgan's laws, and net buffering in one unified transform. The transform is also layout-friendly: it always replaces a gate with one that has the same number of terminals, and preserves routing patterns and placement positions in the design (except for some buffer insertions/deletions). So, although GDM transform is less powerful than resynthesis and remapping, it is more useful in a layout-driven scenario (especially in the later stages), where we wish to maximize circuit performance by minimal layout changes.

We propose a comprehensive GDM transformation algorithm that not only determines the best replacement of a cell along with the best buffering configurations of nets incident on it, but also embeds this into a global scheme for minimizing the delays of large designs. We have successfully applied this transform on large industrial designs built on state-of-the-art industrial libraries. Experiments indicate that one invocation of the GDM transform reduces the circuit delay more than can be obtained by net buffering alone, gate resizing alone, or by a combination of net buffering followed by gate resizing. The GDM transform applied in concert with net buffering and gate resizing also yields the best possible delay results of all combinations of transformations in our repository.

The rest of this paper is organized as follows. In Section 2, we present a summary of previous work on post-technology-mapping delay optimizing transforms. Section 3 introduces terminology and ideas used in the rest of the paper. The central concept of GDM equivalence of cells is described in Section 4. Section 5 presents details of the transform itself: the overall algorithm, GDM-equivalence classes of library cells, and optimizations in computing buffering solutions at the inputs and output nets of the transformed cells. Section 6 demonstrates the effectiveness of the GDM approach vis-a-vis other layout-driven transforms. We comment on some existing implementation inefficiencies in Section 7, and conclude the paper in Section 8.

# 2 Previous Work

## 2.1 Simple Transforms

Under these transforms, the logic functionality of each cell present in the design is not modified. Some simple cells such as buffers may be added or deleted. Examples of such transforms are gate resizing and buffer optimization.

**Gate Resizing** This includes transistor sizing approaches that model area and delay as posynomials [9], linear programming approaches using a piecewise linear delay model [4], convex programming-based approach [18], a cutset-based approach [19] and global sizing – a greedy technique enhanced with a mechanism to get out of local minima [7].

**Buffer Optimization** Work in this category falls into two domains: pre-routing and post-routing. In the pre-routing domain, the problem consists of generating a delay-optimum fanout tree and is known to be computationally hard. A lot of algorithms have been proposed [11, 5, 22, 21, 14, 17] targeting either a single multi-fanout gate and/or an entire design. In the post-global-routing realm, the topology of each net is fixed by the routing tool and the buffer optimization problem reduces to net buffering. Given a net with fixed topology and candidate buffer insertion points, van Ginneken proposed an optimum, polynomial-time, minimum-delay buffer insertion algorithm [16]. Extensions to handle an input-slew based delay model [15], optimal wire segmenting [1], noise [2] and resistive shielding effects [3] have since been proposed.

## 2.2 Complex Transforms

In this class of works, more complex logic transforms are applied on mapped circuits and the logic function of cells may be changed. Examples of such transforms are DeMorgan and general resynthesis & remapping.

In the `LATTIS` system [8], a number of optimizations such as gate resizing, buffer insertion, DEMORGAN, timing directed factorization and remapping are applied sequentially on a technology mapped netlist. The DEMORGAN transform used here replaces a gate with its dual, and adds inverters at the inputs and outputs (if complemented signal is not available). As will be apparent soon, our technique is much more general than DEMORGAN.

Kannan *et al.* [13] and Ishioka [12] proposed post-placement, prerouting optimizations, followed by incremental placement to incorporate the changes. The set of transformations includes fanout optimization and gate resizing applied sequentially [13], and resynthesis and remapping [12]. In [13], the wire-routes are estimated by minimum spanning trees.

Hojat and Villarubia [10] described the results of using simple synthesis operations such as gate sizing between partitioning steps in a mincut based placement algorithm. More recently, Shenoy *et al.* [20] proposed a design flow that incorporates an intermediate iterative step between the traditional synthesis and physical design steps. This step integrates a number of synthesis optimizations in the inner loop of an iterative method to perform global placement under given timing constraints.

One common drawback of all these approaches is that the optimizations are performed before *true* routing information is available. Hence the assumptions regarding global net topology and wire loading and delays, which are crucial in deep sub-micron technologies, may be erroneous.

In [6], Carragher *et al.* proposed a new paradigm in which logic optimization is interleaved with placement/partitioning refinement and hierarchical global routing. The optimization incorporates a comprehensive set of layout-friendly, logic-level transforms for improving the delay and area of a mapped, block-placed, and globally routed circuit under design and technology constraints.

# 3 Preliminaries

Structurally, a *circuit* is an interconnection of cells through nets. An *extended net* is one which passes over buffers and inverters. The source of such a net is either a primary input or the output of a non-buffer and non-inverter cell. Sinks are either primary outputs or inputs of non-buffer and non-inverter cells. For example, suppose the boxes in Fig. 2a represent non-inverter/buffer cells. Then this subcircuit contains three extended nets: (i) with root $G$ and sinks $E$ and $F$, (ii) with root $A$ and sinks $C$ and $I$, and (iii) with root $B$ and sinks $J$ and $D$. Extended nets are the basic objects used in buffering algorithms. Note that in our paradigm, the input to the optimization tool is a fully-mapped, block-placed, and globally-routed design, as shown in Figure 1. Thus, the nets are globally routed and their tree topologies are already known.

The timing analysis is done statically using cell delays and wire delays & loads. Pin-to-pin delays of cells can be computed either by the *load-dependent simple delay model* or *input-slew dependent linear delay model*. In the simple delay model, the delay from input pin $i$ to output pin $o$ of a cell is given by $d_{i,o} = \alpha_{i,o} + \beta_{i,o}.C_o$, where $\alpha_{i,o}$ is the intrinsic delay from $i$ to $o$, $\beta_{i,o}$ is the load coefficient and $C_o$ is the load capacitance visible at output $o$ of the cell. The linear delay model has additional linear terms incorporating the slew at the input pin. Since the global routing has already been performed, actual wire loads and delays can be computed. Currently, we use Elmore delay model for computing the wire delays.

The *arrival time* at a net is the time at which a signal propagating from the primary inputs reaches the net. The *required time* is the time by which the signal should reach the net for the circuit to meet timing requirements. The *slack* is the difference between the required time and the arrival time.

Given a globally routed extended net with required times and desired signal phases (even/odd inversions from source) at all sink nodes, an optimal net buffering algorithm chooses buffers and inverters from a library and inserts them at appropriate places on the extended net, such that all sink phase requirements are satisfied and the required time at the root is maximized. If the root is an output pin of a cell, the algorithm doesn't stop at maximizing the slack at the root; instead it chooses the buffering solution that maximizes the minimum slack at the inputs of the cell. In this paper, we assume that such an algorithm (van Ginneken's [16] or its extension by Lillis *et al.* [15]) is available to us.

# 4 GDM-Equivalence of Functions

Two boolean functions, $f$ and $g$, are said to be *NN-equivalent* or *GDM-equivalent* if (i) both have the same support set, and (ii) either $f$ or $f'$ can be obtained by complementing zero or more inputs of $g$. For example, let $f(x_1, x_2) = x_1 + x_2$ and $g(x_1, x_2) = x_1 x_2$. Since $f'(x_1, x_2) = g(x_1', x_2')$, functions $f$ and $g$ are GDM-equivalent.

More generally, let $\mathcal{F}(x_1, x_2 \ldots x_n)$ be a set of $m$ functions $\{f_1, f_2 \ldots f_m\}$. The support set of each $f_i$ is assumed to be a subset of $\{x_1, x_2 \ldots x_n\}$. Let $I$ be an $(n+m)-$dimensional binary vector that encodes the complementation of inputs and outputs of $\mathcal{F}$ as follows. If the $i^{th}$ bit of $I$ (henceforth $I[i]$) is 1 and $1 \leq i \leq n$, then input $x_i$ of $\mathcal{F}$ is complemented; otherwise, it is left uncomplemented. Similarly, if $I[n+i]$ is 1 and $1 \leq i \leq m$, the $i^{th}$ output, $f_i$, of $\mathcal{F}$ is complemented; otherwise it is not complemented. Vector $I$ is called the *inversion vector* of $F$, and the final set of functions obtained after complementing inputs and outputs of $F$ according to $I$ is denoted

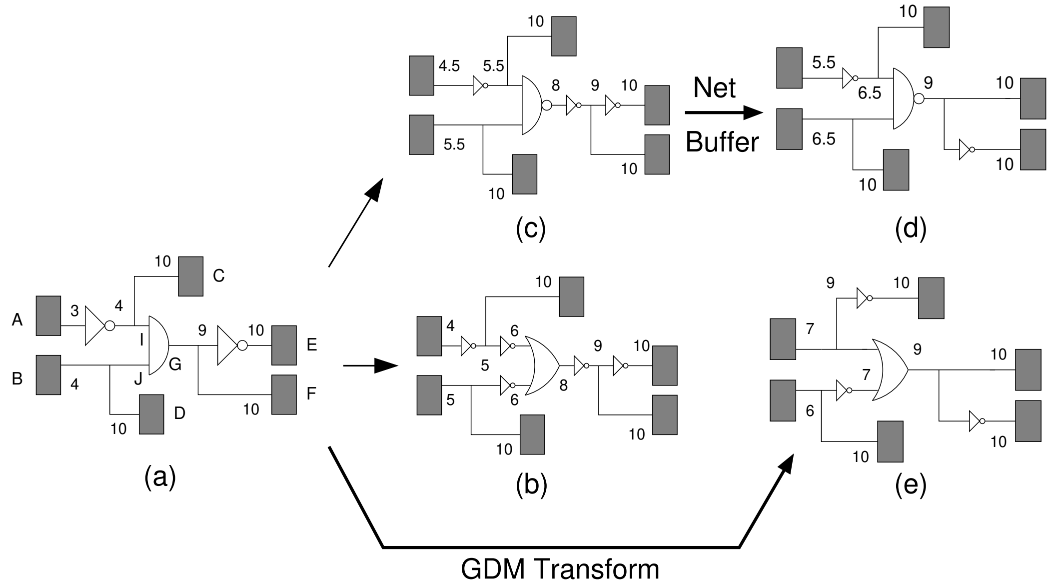


Figure 2: Comparison of GDM transformation with simple approach.

$\mathcal{F}_I$. As an example, let $\mathcal{F}(x_1, x_2) = \{x_1x_2, x_1\}$ and $I = [0, 1, 1, 0]$. Then $\mathcal{F}_I(x_1, x_2) = \{(x_1x_2')', x_1\}$ or $\{x_1' + x_2, x_1\}$.

Using the above notation, two sets of functions $\mathcal{F}(x_1 \ldots x_n)$ and $\mathcal{G}(x_1 \ldots x_n)$ are said to be GDM-equivalent if both have the same number $(m)$ of functions, and there exists an inversion vector $I$ of $\mathcal{G}$ such that the $i^{th}$ function of $\mathcal{G}_I(x_1 \ldots x_n)$ is identical to $f_i$ for all $i$ in 1 to $m$.

# 5 The GDM Transform

*GDM transformation* of a cell $F$ (multi-output in general) essentially involves replacing $F$ by a GDM-equivalent cell $G$, and inserting and/or deleting buffers and inverters on the extended nets at the inputs and outputs of $G$ to preserve the functionality of $F$.

In a simplistic approach, one would use capacitance and delay information from the current buffering scheme on the extended input and output nets of $\mathcal{F}$ to determine the best GDM-equivalent cell $\mathcal{G}$ for replacing $\mathcal{F}$. Thereafter, inverters would be added at the appropriate inputs and outputs of $\mathcal{G}$ to make it functionally equivalent to $\mathcal{F}$. Finally, optimal net buffering would be applied to these nets to determine the best distribution of buffers and inverters, *for the already determined choice of replacement cell* $\mathcal{G}$. The drawback of this approach is that it fails to take into account the effect (capacitance and delays) of the final net buffering solutions when determining which replacement cell $\mathcal{G}$ to use for $\mathcal{F}$. Consequently, the optimal combination of replacement cell and net buffering solutions may be missed. The innovation in our approach lies in applying optimal net buffering to the input and output nets of the cell to implement inversions mandated by each choice of replacement cell $\mathcal{G}$, and then using this information to choose the best combination of $\mathcal{G}$ and input and output buffering. Thus, information from the final buffering scheme is used to guide the best choice of $\mathcal{G}$. While it may appear that computing buffering solutions for each input and output extended net for each choice of $\mathcal{G}$ is prohibitively expensive, the computation of buffering solutions can be significantly optimized, as described in Section 5.3.

To illustrate the advantage of the GDM transformation vis-a-vis choosing replacement cell $\mathcal{G}$ first and then buffering input and output nets, consider the subcircuit shown in Fig 2a. Here, each shaded box represents a non-inverter/buffer cell sharing an extended net with the AND gate. The numbers represent required times at the corresponding points in the circuit. We wish to replace the AND gate with a GDM-equivalent cell and buffer its inputs and outputs such that the minimum required time at $A$ and $B$ is maximized. For simplicity, let us assume that the cell library has only 4 cells: inverters with a fixed delay of 1, two-input AND gates with a fixed delay of 5 from each input to output, two-input NAND gates with a fixed delay of 2.5 from each input to output, and two-input OR gates with a fixed delay of 2 from each input to output.

If we choose to replace the AND gate first, and then insert inverters at the inputs and/or outputs, we have two choices as shown in Figs. 2b and c. Since the minimum required time at $A$ and $B$ in Fig. 2c is higher than that in Fig. 2b, the NAND gate is chosen as the replacement for the AND gate. Subsequently, on applying net buffering techniques to the input and output nets of the NAND gate, the subcircuit in Fig. 2d results. Here, the minimum required time at $A$ and $B$ is 5.5. On the other hand, if we apply the GDM transformation, the subcircuit shown in Fig. 2e is produced. The minimum required time at $A$ and $B$ in this subcircuit is 6, so this is a better transformation than Fig. 2d. Note that the GDM transformation considers the effect of buffering the input and output nets when determining the best choice for the replacement cell, and therefore, is able to identify that the subcircuit in Fig. 2b eventually leads to a better subcircuit (Fig. 2e) than the subcircuit eventually obtainable from Fig. 2c (i.e., Fig. 2d).

It is important, though easy, to see that gate sizing, net buffering, and simple DeMorgan transform (i.e., replacing a gate with its dual, such as an AND with an OR) are special cases of this transformation. For instance, gate sizing is subsumed in the GDM transform, since a function $f$ is GDM-equivalent to itself (by definition) and hence the GDM transform automatically considers replacing a cell $\mathcal{F}$ by another cell $\mathcal{G}$ that has the same function as $\mathcal{F}$ but has a different size. However, our current GDM implementation imposes some restrictions; see the remarks around Figure 4.

## 5.1 Basic Algorithm

The basic algorithm for choosing the best GDM transform of a cell $F$ for timing optimization is shown in Fig 3. Note that inverters and buffers are not referred to as cells in this discussion.

Note that function GDM_cell does not actually *implement* any transformation; it simply *determines* the one that maximizes the *minimum slack at the inputs of cells driving* $F$ (henceforth, local minimum slack). The procedure for applying GDM transformations globally to the entire circuit is divided into 3 phases.

(a) **Evaluation:** Let $C$ be the set of all critical cells in the circuit.

**GDM_cell** (circuit $K$, cell $F$, library $L$)
Let $In$ : set of cells/primary inputs driving extended input nets of $F$.
$Out$ : set of cells/primary outputs driven by extended output nets of $F$.
$s_F$ : minimum slack at input pins of cells in $In$.
$E$ : set of cells in $L$ that are GDM-equivalent to $F$.

1. Save current buffering of input and output extended nets of $F$.
2. For each cell $G$ in $E$
   (a) Replace $F$ by $G$
   (b) $I \leftarrow$ inversion vector to make $G$ equivalent to $F$.
   (c) Apply optimal net buffering routine to input and output extended nets of $G$ to implement inversions encoded in $I$.
   (d) Propagate required times from cells in $Out$ to those in $In$ using delay parameters of $G$, and the buffering solutions of step 2(c)
   (e) $s_G \leftarrow$ new min slack at the inputs of cells in $In$
3. Restore cell $F$ and the original buffering of input and output extended nets.
4. Let $G^*$ be the replacement cell that maximizes $s_G$, and let $N^*$ be the corresponding buffering solutions for extended input and output nets of $G^*$.
5. If $s_{G^*} > s_F$, return $(G^*, N^*)$
   else return ($F$, original net buffering).

Figure 3: Finding best GDM transform of a cell.

For each cell $F$ in $C$, the best GDM transformation is determined by invoking function GDM_cell. Each transformation $(G^*, N^*)$ thus obtained is assigned a cost, based on (i) local min slack improvement $(s_{G^*} - s_F)$ in function GDM_cell, and (ii) area penalty of replacing the original cell $F$ by $G^*$ and due to implementation of buffering solution $N^*$. Let $T$ be the set of best GDM transformations for all cells in $C$. The set $T$ is kept sorted in increasing order of the cost of transformations.

**(b) Selective implementation:** Given the sorted set $T$, the transformation with the least cost is first implemented. Let this be the transformation $(G^*, N^*)$ for critical cell $F$ in the original circuit. Now, implementing $N^*$ potentially modifies the buffering of some input and output extended nets of $F$. So, if there exists another critical cell $F'$ that shares an input or output extended net with $F$, then there can be a potential conflict between the buffering solutions for the shared net in the GDM transformations for $F$ and $F'$. Thus, once the GDM transformation for $F$ is implemented, the already-computed GDM transformation for $F'$ may be rendered sub-optimal or even invalid (e.g., if they differ in the phases of signals at input or output pins of $G^*$). Therefore, the GDM transformations of all critical cells that share an extended net with $F$ are removed from the set $T$ after $(G^*, N^*)$ is implemented.[1] Of the remaining transformations in $T$, the one with the lowest cost is then chosen for implementation and the same process repeated until the set $T$ is empty.

As an example, suppose cells $A$, $B$, $C$, $D$, $E$ and $F$ in Fig 4 are critical. Initially, the best GDM transformations for all cells are determined, and suppose the cost of the transformations increases from $A$ to $F$. Using the above strategy, once the transformation for $A$ is implemented, those for $B$, $E$ and $F$ are invalidated because they share some extended net with $A$. The transformation for $C$ is thus implemented next. This, in turn, invalidates the transformation for $D$. Therefore, only $A$ and $C$ are transformed. Note that due to this invalidation policy, the current GDM implementation may not strictly subsume our gate resizing or net buffering implementation. Our gate resizing (net buffering) algorithm can potentially resize any gate (buffer any nets), whereas a given iteration of GDM will not simultaneously resize any two gates that share extended nets (buffer two nets $N_1$ and $N_2$ whose root cells share either $N_1$ or $N_2$).

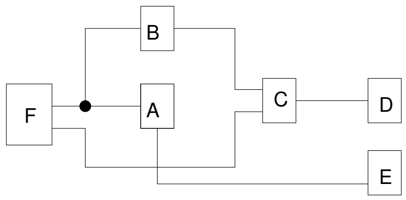


Figure 4: Illustrating sharing of extended nets by critical cells.

**(c) Partial undoing:** Once the set $T$ becomes empty, a static timing analyzer is run on the modified circuit to determine the new minimum slack. If this is larger than the original minimum slack $s_0$ (before the transformations were applied), the new set $C$ of critical cells is determined and phases (a), (b) and (c) repeated. However, if it is determined that the minimum slack has worsened[2], we undo the transformations in small groups in the following manner. We first undo a fixed small number (10 in our experiments) of the costliest transformations and re-evaluate the minimum slack in the circuit. If the minimum slack increases beyond the original slack $s_0$, we accept the remaining transformations (not yet undone), and repeat phases (a), (b) and (c). Otherwise, we undo a small number of the remaining costliest transformations, and continue the process until either the min slack increases beyond $s_0$ or all transformations are undone. If all transformations are undone, we terminate the iterative process.

Note that in phase (b), one can choose to re-compute the best GDM transformations of all cells on a shared extended net whenever one of the gates is transformed. However, this entails significant computation every time a transformation is implemented, and most of the results re-computed are invalidated again soon. Moreover, given the inevitable inaccuracies in estimating wire delays and capacitances at this stage of the design flow, we believe that it is not worth the effort. Our experiments also suggest that even if the GDM transformation for a cell $G$ is invalidated in an iteration, there is a fair chance of $G$ being reconsidered (after re-evaluation of its best GDM transformation) in a later iteration if $G$ continues to remain critical.

While the basic algorithm for timing optimization of a circuit by GDM transformation is as given above, there are several important issues related to implementation. In the following subsections, we address these briefly.

## 5.2 Equivalence Classes of Library Cells

GDM-equivalence is easily seen to be reflexive, symmetric and transitive. Therefore, GDM-equivalence is an equivalence relation and can be used to partition the cells in a library into disjoint equivalence classes. The motivation for creating equivalence classes comes from the observation that once they are created, determining the set $E$ in function GDM_cell (see Fig. 3) reduces to accessing the equivalence class of $F$. Since the number of critical cells in real designs often runs into thousands, it is beneficial to compute the equivalence classes once and reuse the results.

In order to determine all cells that are GDM-equivalent to $\mathcal{F}$, we proceed by filtering out as many cells as possible by a series of

[1] As an optimization, one can check if the buffering solutions are in conflict, and invalidate the GDM transformation for $F'$ only if a conflict exists. For practical reasons related to the difficulty of checking if two buffering solutions of an extended net are in conflict, we choose to bypass this check and assume a conflict. Our experience suggests that this doesn't degrade significantly the quality of the final solution.

[2] Although each GDM transformation individually doesn't worsen the slack, interactions between multiple transformations can cause a degradation.

simple tests. These tests either filter out a cell $\mathcal{G}$, or yield information about values of bit positions in the inversion vector $I$ of $\mathcal{G}$. For the remaining bit positions, we need to assign all possible combinations of bit values to obtain the possible inversion vectors. For each vector $I$ thus generated, we then use BDDs to check the equivalence of each function $f_i$ in $\mathcal{F}$ with the corresponding function of $\mathcal{G}_I$. If an input vector $I$ is obtained such that the BDD equivalence checks succeed for all functions $f_i$ of $\mathcal{F}$, we put $\mathcal{G}$ in the same equivalence class as $\mathcal{F}$.

Below, we list some simple tests to eliminate a candidate cell G from the equivalence class of $\mathcal{F}$.

1. If the number of inputs and outputs of $\mathcal{G}$ and $\mathcal{F}$ don't match, $\mathcal{G}$ is not GDM-equivalent to $\mathcal{F}$.
2. Let there be $n$ inputs and $m$ outputs of each of $\mathcal{G}$ and $\mathcal{F}$. Let $N(g_i)$ be the onset count (number of minterms) of function $g_i$ of $\mathcal{G}$ and let $N(f_i)$ be the onset count of the corresponding function $f_i$ of $\mathcal{F}$ (correspondence is by index number). The onset counts can be easily determined once BDDs of the functions in $\mathcal{G}$ and $\mathcal{F}$ are constructed. If $N(g_i) \neq N(f_i)$, then $g_i$ is not equivalent to $f_i$. In addition, if $N(g_i) \neq 2^n - N(f_i)$, then $g_i$ is not equivalent to $f_i'$ either. Therefore, if both tests are satisfied, $\mathcal{G}$ is not GDM-equivalent to $\mathcal{F}$.
3. Let $g_i|_a$ be the cofactor of function $g_i$ of $\mathcal{G}$ with respect to input $a$. Using the notation for onset counts, If $N(g_i|_a) \neq N(f_i|_a)$ or $N(g_i|_{a'}) \neq N(f_i|_{a'})$, then $g_i$ and $f_i$ cannot be equivalent with the same phase of input $a$ applied to both. Similarly, if $N(g_1|_a) \neq N(f_i|_{a'})$ or $N(g_1|_a) \neq N(f_i|_{a'})$, then $g_i$ and $f_i$ cannot be equivalent with opposite phases of input $a$ applied to $f_i$ and $g_i$. In a similar manner, we can check from the onset counts if it is possible for $g_i$ and $f_i'$ to be equivalent with same (opposite) phase of input $a$ applied to both. If all the above tests fail (i.e., neither $f_i$ nor $f_i'$ can be equivalent to $g_i$), we infer that $f_i$ is not GDM-equivalent to $g_i$; so $\mathcal{F}$ and $\mathcal{G}$ are not GDM-equivalent.

When the tests involving onset counts fail to eliminate $\mathcal{G}$ from the equivalence class of $\mathcal{F}$, they usually provide useful information about the inversion vector $I$ of $\mathcal{G}$. For example, suppose $N(g_i) = N(f_i)$ and $N(g_i) \neq 2^n - N(f_i)$. It's easy to infer from this that if $\mathcal{G}$ and $\mathcal{F}$ are GDM-equivalent, their $i^{th}$ output must be in the same phase, i.e., the $n + i^{th}$ bit in the inversion vector must be 0. Now, suppose another test (using onset counts of co-factors), indicates that $g_i$ and $f_i$ must be in opposite phase if $\mathcal{F}$ and $\mathcal{G}$ are GDM-equivalent. Since we have a contradiction, we can then conclude that $\mathcal{G}$ is not GDM-equivalent to $\mathcal{F}$. In other cases, when no such contradictions arise, the information about bit values in the inversion vector helps reduce the number of BDD equivalence checks that need to be finally performed. For example, suppose in an 8-bit inversion vector $I$, we have already inferred the values of 4 bits from tests involving onset counts. In order to complete our test of whether $\mathcal{G}$ is GDM-equivalent to $\mathcal{F}$, we need to check if the BDD for each $f_i$ in $\mathcal{F}$ is equivalent to the BDD for the corresponding function in $\mathcal{G}_I$ for one out of $2^4 = 16$ combinations of unknown bit values in $I$. In contrast, if we do not have any knowledge of bit values in $I$, we must consider $2^8 = 256$ combinations.

Our method for identifying GDM-equivalence classes is able to identify interesting cases where a gate is GDM-equivalent to another gate of the same functionality, but with some inputs and/or outputs inverted. For example a two-input XOR is equivalent to another two-input XOR with either (i) no inputs or outputs inverted, or (ii) one input and output inverted, or (iii) both inputs inverted. Such equivalences cannot be identified by gate sizing or net-buffering alone, but can prove useful when trying to optimize a design with minimal layout changes.

### 5.3 Buffering Extended Nets at Inputs and Outputs

We saw in Section 5.1 that for each candidate cell $\mathcal{G}$ that is GDM-equivalent to $\mathcal{F}$, we must compute optimal net buffering solutions for extended nets at the inputs and outputs of $\mathcal{G}$. However, recomputing the solutions from scratch each time is very time consuming and wasteful. By carefully analyzing the problem, it is possible to store partial solutions of the buffering problems at suitable points on the extended nets and reuse them for all candidate cells $\mathcal{G}$. This leads to significant run-time improvements of the GDM transformation.

Let us first consider extended nets at cell outputs. As an example, suppose the shaded cell in Fig. 5 is being GDM transformed. The extended net from output $O$ goes to sink pins $S_1$ and $S_2$ of other cells. Clearly, GDM transformation of the shaded cell does not affect the required times or phases of signals[3] at $S_1$ and $S_2$ under the load-dependent delay model. The optimal buffering solution for this extended net, as computed by van Ginneken's [16] algorithm (with Lillis' [15] extensions) depends only on the required times and signal phases at the sink nodes, and hence is independent of the replacement cell used for the shaded cell. The buffering algorithm essentially generates two solutions: one which expects an inverted signal from the source of the net, and the other which expects the driving signal to be non-inverted. The best solution is then obtained by choosing the one that maximizes the required time at the source. This solution strategy for the buffering problem fits perfectly with our requirements, since some GDM-equivalent cells require inversion of the output before driving the extended net, while others don't. Thus, instead of throwing away one solution, we can store the buffering solutions for both phases of the output at the root node $O$ of the extended net. This essentially amounts to stopping short of the final step in van Ginneken's algorithm. Depending on whether an inversion is mandated on the output by the GDM-equivalent cell or not, we can then choose the appropriate solution from the stored pair of solutions. This effectively reduces the number of computations of buffering solutions for each extended net at the output to only once.

Now let us look at extended nets at the inputs of the cell. In Fig. 5a, the net from $D$ to sink nodes $I_1$, $I_2$, $S_3$, $S_4$, $S_5$ and $S_6$ represents one such net. In this case, the required times and phases at sink nodes $I_1$ and $I_2$ can change as a result of GDM transformation of the shaded cell. However, the required times and phases of $S_3$, $S_4$, $S_5$ and $S_6$ do not depend on the transformation of the shaded cell. To deal with this situation, we need to examine van Ginneken's optimal net buffering algorithm [16] more closely. In his algorithm, solutions are first computed at each sink node of a net, and then they are propagated towards the root node by merging solutions from the children nodes at the parent Steiner node. So, in our example, solutions from $S_3$ and $S_4$ can be merged at Steiner node $T_1$, and similarly solutions from $S_5$ and $S_6$ can be merged at $T_2$, regardless of which cell is used to replace the shaded cell. However, the buffering solutions from $I_1$ and $I_2$ depend on the required time and signal phases at $I_1$ and $I_2$, which in turn, depend on the replacement cell and the corresponding output buffering solutions. Therefore, we cannot propagate solutions from $I_1$ and $I_2$ to $T_3$. This, in turn, prevents solutions at $T_1$ from being propagated to $T_4$ and solutions at $T_2$ from being propagated to $T_0$. Thus, we cannot compute the buffering solution for the complete extended net independent of the choice of the replacement cell. Nevertheless, we can save some recomputation by storing partial results that do not depend on the choice of the replacement cell at intermediate nodes such as $T_1$ and $T_2$. These partial results can then be used to compute complete buffering solutions for the input extended nets for all candidate replacement cells. In our example, once a choice of the replacement cell and the corresponding output buffering solution is known, the required times and capacitances at nodes $I_1$ and $I_2$ are computed and the solutions propagated to $T_3$ and then to $T_0$. Note that the amount of computational savings in this case depends on the net topology. For example, if the extended net at the input had the topology shown in Fig. 5b, we can propagate the solutions from $S_3$, $S_4$, $S_5$ and $S_6$ to node $T_4$. Thus, we would need to perform 4 computation/merging of input buffering solutions (at $I_1$, $I_2$, $T_3$ and $T_0$) for every choice of the replacement cell. In contrast, the number of computation/merging for the corresponding net in Fig. 5a is 5 per

[3]Recall that after GDM transformation, the functionality of the original cell is preserved.

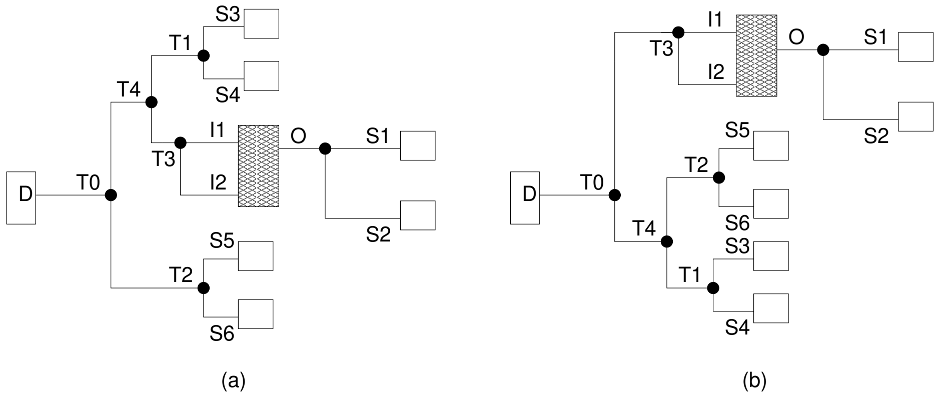


Figure 5: Extended nets to be buffered in GDM transformation.

| ex | #gates | #nets | gate area (in BC) | orig-delay (in ns) |
|---|---|---|---|---|
| ex1 | 356 | 409 | 1567 | 7.84 |
| ex2 | 17.1K | 26.0K | 122.6K | 7.44 |
| ex3 | 32.0K | 37.7K | 343.6K | 14.49 |
| ex4 | 40.0K | 48.1K | 200.2K | 11.38 |
| ex5 | 86.7K | 108.1K | 381.6K | 18.41 |
| ex6 | 172.2K | 210.9K | 718.6K | 56.36 |

1K = 1000, 1 BC = area of smallest inverter in the library. All benchmarks are in 0.35-$\mu$ technology.

Table 1: Benchmark Statistics

GDM transformation.

# 6 Experimental Results

To test the effectiveness of the GDM transform, we used six optimized, mapped, block-placed, and globally routed real industrial designs. Table 1 shows the relevant statistics of these designs such as the number of gates (a gate could be as simple as an inverter or as complex as an 8-bit adder), number of simple nets, the total gate area of the design (in terms of the smallest inverter in the library), and the original delay (i.e., delay before any optimization by our tool) of the mapped, block-placed, and globally-routed circuit. This delay takes into consideration pin-to-pin delays through the cells (the input slew-based linear delay model is used) and the wire loads and delays (we use Elmore delays). The two largest designs, ex5 and ex6, are *hi-vision TV encoder/decoder* designs. ex6 has 172K gates and 211K simple nets. The experiments were done on an Ultrasparc 60 with 768MB RAM.

We used the following set-up. To check the power of the GDM transform, we compare it with net buffering (nb), with gate resizing (gs), and with a composed transform consisting of net buffering followed by gate resizing (nb + gs). The results are shown in Table 2. It can be seen that GDM is much more effective in reducing the design delay than nb or gs on all designs, except ex2 on which it yields the same delay, and ex3 on which it is marginally worse than nb, although much better than gs. The worse performance of GDM as compared to nb can be attributed to the following fact. As we mentioned in Section 5.1, our GDM implementation does not strictly subsume gate resizing or net buffering, because it prohibits simultaneous resizing of any gates that share nets or simultaneous buffering of two nets $N_1$ and $N_2$ whose root cells share either $N_1$ or $N_2$. The effectiveness of GDM vis-a-vis nb or gs can be correlated with the number of cell replacements which require inversions at the input or output nets of the cell (column #dm). On ex2 and ex3, the number of such cells is very small: 1 and 3 respectively, and GDM is not so effective. However, on ex4, ex5, and ex6, this number is much higher, resulting in much better delay improvement for GDM vis-a-vis nb and gs.

Also, GDM is more powerful than a combination of nb and gs in three out of six designs, equally powerful in one, and less powerful in the remaining two.

Finally, to see the combined effect of all transforms, we applied nb followed by gs followed by GDM (we found that it is better than the order GDM + nb + gs). It can be seen that this achieves better delay results than nb + gs or GDM alone. The average delay improvement over nb + gs is 3.2%, with the best improvement being 7% for ex1 and ex5. Although this may seem a small improvement, it is very significant for the following two reasons:

1. Today in industry, net buffering and gate resizing are the most popular in-place optimization techniques used during layout. However, if the circuit does not meet the timing constraints after an application of these techniques, the designer is left with no real choice except to painstakingly resynthesize, remap, replace, and re-route parts of the design. As can be seen from Table 2, the GDM transform is an additional in-place optimization that can come to the rescue of the designer if the traditional techniques fail to meet the timing and the extra 4-5% delay improvement may be enough to meet the timing.
2. Since the design is close to the final layout, this timing improvement translates to real improvement in the final circuit delay.

Of course, there is a run-time penalty associated with GDM. As compared to nb + gs, GDM takes 66% more CPU time on average. Also, nb + gs + GDM takes 53% more time than nb + gs. In the next section, we describe one solution to reduce the run-time.

# 7 Excessive Buffering Penalty

In some circuits, such as ex4 in Table 2, our current implementation of GDM can cause excessive area penalty. Consider a cell $\mathcal{F}$ that is being GDM-transformed. Suppose the replacement cell for $\mathcal{F}$ is $\mathcal{G}$. The current implementation of the algorithm attempts to optimally insert buffers and/or inverters for reducing the delay on every input and output extended net of $\mathcal{G}$, regardless of whether the net is critical or not. Since delay optimization is usually area-hungry, this leads to a large number of buffers being inserted on several non-critical input and output nets and thus a large area penalty. Indeed, the buffering algorithm for a net determines the best delay-buffering for the net regardless of whether it is critical or not. So it is the onus of the GDM algorithm to present only critical nets to the delay-reducing buffering algorithm; the remaining nets should be buffered, if needed (for polarity fixing), using an area-aware delay-constrained algorithm, such as [15]. This optimization is currently being developed in our tool. We expect the area penalty due to buffering of input and output

| ex | nb | | | gs | | | nb + gs | | | GDM | | | | nb + gs + GDM | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | n.d. (ns) | $\Delta A$ (BC) | CPU (sec) | n.d. (ns) | $\Delta A$ (BC) | CPU (sec) | n.d. (ns) | $\Delta A$ (BC) | CPU (sec) | n.d. (ns) | $\Delta A$ (BC) | #dm | CPU (sec) | n.d. (ns) | $\Delta A$ (BC) | CPU (sec) |
| ex1 | 4.84 | 96 | 4 | 5.16 | 187 | 6 | 4.84 | 265 | 5 | 4.49 | 201 | 6 | 18 | 4.49 | 352 | 16 |
| ex2 | 7.32 | 70 | 58 | 7.32 | 112 | 80 | 7.32 | 70 | 67 | 7.32 | 78 | 1 | 67 | 7.32 | 70 | 73 |
| ex3 | 10.50 | 436 | 199 | 12.43 | 97 | 142 | 10.42 | 775 | 260 | 10.61 | 417 | 3 | 174 | 10.42 | 775 | 288 |
| ex4 | 9.11 | 1075 | 214 | 9.03 | 676 | 202 | 9.11 | 1644 | 258 | 8.77 | 4653 | 61 | 392 | 8.75 | 1757 | 285 |
| ex5 | 10.50 | 622 | 485 | 12.13 | 1157 | 371 | 10.24 | 3552 | 673 | 9.70 | 3001 | 104 | 1133 | 9.53 | 4302 | 934 |
| ex6 | 43.00 | 2058 | 1444 | 48.62 | 583 | 1479 | 40.15 | 2604 | 1679 | 40.98 | 2508 | 48 | 2133 | 39.79 | 3004 | 2222 |

nb = net buffer, gs = gate size, nb + gs = nb followed by gs, GDM = generalized DeMorgan transform.
n.d. = new delay (after optimization), $\Delta A$ = area penalty. CPU time includes time to read the circuit.
#dm = number of cell replacements involving at least one inversion at input/output nets of the cell.

Table 2: Experimental Results

nets to reduce considerably as a result of this optimization. We also expect huge speed-up in the run-time of the transformation, since we skip computing buffering solutions of non-critical nets on which the source and sink phases remain unchanged.

# 8 Conclusions

In this paper, we have proposed a new layout-driven, timing-oriented logic optimization technique Generalized DeMorgan (GDM) transform. It is a powerful transform for optimizing circuits in the post-global-route-and-place, pre-detailed-route-and-place stage of the design flow. At this stage of the design flow, maximal improvements in performance with minimal changes to the circuit configuration and layout are desirable. The GDM transform has exactly this property. It replaces one cell by a GDM-equivalent cell with the same number of inputs and outputs, and simply inserts and/or deletes buffers and inverters on the extended input and output nets. Consequently, the placement and routing of cells is minimally disturbed.

Gate sizing, net buffering and simple DeMorgan transform are special cases of this transformation. Thus, the GDM transformation tightly integrates gate sizing, net buffering and replacing a gate by its GDM-equivalent, all in one. Our experimental results indicate that one pass of GDM transformation of a circuit produces delays that are better than those obtained by gate sizing and net buffering applied in concert. Although GDM is less powerful than arbitrary resynthesis and remapping, it is more layout-friendly and therefore of greater practical utility in a layout-driven framework.

We believe that powerful yet layout-friendly transforms like the GDM transform are necessary steps in the search for a predictable and acceptable solution of the timing convergence problem.

# References


[1] C. J. Alpert and A. Devgan. Wire Segmenting For Improved Buffer Insertion. In *DAC*, pages 588–593, 1997.

[2] C. J. Alpert, A. Devgan, and S. T. Quay. Buffer Insertion for Noise and Delay Optimization. In *DAC*, pages 362–367, 1998.

[3] C. J. Alpert, A. Devgan, and S. T. Quay. Buffer Insertion With Accurate Gate and Interconnect Delay Computation. In *DAC*, pages 479–484, 1999.

[4] M. Berkelaar and J. Jess. Gate Sizing in MOS Digital Circuits with Linear Programming. In *The Proceedings of the European Conference on Design Automation*, pages 217–221, 1990.

[5] C. L. Berman, J. L. Carter, and K. F. Day. The Fanout Problem: From Theory to Practice. In C. L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pages 69–99. MIT Press, March 1989.

[6] R. Carragher, R. Murgai, S. Chakraborty, M. Prasad, A. Srivastava, and N. Vemuri. Layout-driven Logic Optimization. In *IWLS*, pages 270–276, 2000.

[7] O. Coudert, R. Haddad, and S. Manne. New Algorithms for Gate Sizing: A Comparative Study. In *DAC*, pages 734–739, 1996.

[8] J. P. Fishburn. LATTIS: An Iterative Speedup Heuristic for Mapped Logic. In *DAC*, pages 488–491, 1992.

[9] J. P. Fishburn and A. E. Dunlop. TILOS: A Posynomial Programming Approach to Transistor Sizing. In *ICCAD*, pages 326–328, 1985.

[10] S. Hojat and P. Villarubia. An Integrated Placement and Synthesis Approach for Timing Closure of PowerPC Microprocessor. In *ICCD*, pages 206–210, 1997.

[11] H. J. Hoover, M. M. Klawe, and N. J. Pippenger. Bounding Fan-out in Logical Networks. *Jour. of the Association for Computing Machinery*, 31(1):13–18, January 1984.

[12] T. Ishioka, M. Murofushi, and M. Murakata. Layout Driven Delay Optimization With Logic Re-synthesis. In *IWLS*, 1997.

[13] L. Kannan, P. Suaris, and H. G. Fang. A Methodology and Algorithms for Post-Placement Delay Optimization. In *DAC*, pages 327–332, 1994.

[14] D. Kung. A Fast Fanout Optimization Algorithm for Near-Continuous Buffer Libraries. In *DAC*, pages 352–355, 1998.

[15] J. Lillis, C. K. Cheng, and T. T. Y. Lin. Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model. In *ICCAD*, pages 138–143, 1995.

[16] Lukas P. P. P. van Ginneken. Buffer Placement in Distributed RC-tree Networks for Minimum Elmore Delay. In *ISCAS*, pages 865–868, 1990.

[17] P. Rezvani, A. Ajami, M. Pedram, and H. Savoj. LEOPARD: A Logical Effort-based Fanout OPtimizer for ARea and Delay. In *ICCAD*, pages 516–519, November 1999.

[18] S. S. Sapatnekar, V. B. Rao, P. M. Vaidya, and S. M. Kang. An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization. In *IEEE Transactions on Computer-Aided Design*, pages 1621–1634, November 1993.

[19] H. Savoj, K. Xiang, K. Pan, and A. Domic. Technology Dependent Timing Optimization. In *IWLS*, 1997.

[20] N. Shenoy, M. Iyer, R. Damiano, K. Harer, H-K. Ma, and P. Thilking. A Robust Solution to the Timing Convergence Problem in High Performance Designs. In *ICCD*, pages 250–257, Oct. 1999.

[21] K. J. Singh. *Performance Optimization of Digital Circuits*. PhD thesis, UC Berkeley, December 1992.

[22] H. Touati. *Performance-oriented Technology Mapping*. PhD thesis, UC Berkeley, November 1990. UCB/ERL M90/109.