Computing the Entire Active Area / Power Consumption versus Delay Trade–off Curve for Gate Sizing with a Piecewise Linear Simulator

Michel R.C.M. Berkelaar[1,2], Pim H.W. Buurman[2] and Jochen A.G. Jess[2]

[1]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
[2]Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

The gate sizing problem is the problem of finding load drive capabilities for all gates in a given Boolean network such, that a given delay limit is kept, and the necessary cost in terms of active area usage and / or power consumption is minimal. This paper describes a way to obtain the entire cost versus delay trade–off curve of a combinational logic circuit in an efficient way. Every point on the resulting curve is the global optimum of the corresponding gate sizing problem. The problem is solved by mapping it onto piecewise linear models in such a way, that a piecewise linear (circuit) simulator can do the job. It is shown that this setup is very efficient, and can produce trade–off curves for large circuits (thousands of gates) in a few minutes. Benchmark results for the entire set of MCNC '91 two–level examples are given.

1 Introduction

The problem treated in this paper is the problem of gate sizing. It can be defined as assigning load drive capabilities to the gates in a Boolean network such, that a given delay limit is obeyed, and the total cost in terms of active area and / or power consumption of the circuit is minimal. The problem is very similar to the transistor sizing problem. The main difference is that in gate sizing all transistors in a logic gate are sized simultaneously, whereas transistor sizing sizes single transistors. The main reason we focus on gate sizing is the fact that we want to be able to optimize large circuits. Our ideas are, however, also applicable to transistor sizing.

Previous work

In the past, several algorithms have been proposed to solve the transistor sizing problem. There are a number of heuristic or combined algorithmic/heuristic optimizers [15] [9] [8] [10] [6] [18] which solve the problem but cannot guarantee optimality of the solution. More recently a number of solutions which use efficient forms of solving a nonlinear programming problem [11] [12] [16] [17] have been published. They prove that in many cases the heuristic solutions stay far from the global optimum. The most important problems in these nonlinear programming approaches are usually the run time, which becomes too long for large examples, and the convergence. In [1] and [3], a solution to the gate sizing problem was proposed using linear programming (LP), and piecewise linear (PL) approximations of the nonlinear delay formulas. This proved to be fast and, therefore, feasible for large circuits.

Why compute the entire trade–off curve?

All of the above approaches can find one point in the solution space of the problem per invocation of the program. Very often, however, a designer is interested in (part of) the trade–off curve, to be able to compare the advantages and disadvantages of several possible implementations. If we examine typical trade–off curves as in Figure 3 it is immediately apparent that most designers will want to avoid using solutions in the leftmost parts of the curves, where small gains are made at large costs.

To obtain the trade–off curves, one could calculate a lot of points by repeatedly using a single–point optimization algorithm. This is not only computationally expensive, but it is also not easy to determine which points are needed to make linear interpolation between them an accurate estimation of the real trade–off curve. It is just this task that our circuit simulator with variable integration step size performs very efficiently. By introducing a time–dependent delay, (part of) the trade–off curve is visited during the simulation. Because the different solutions for two slightly different delays are usually close to each other, the new solution can be calculated with only a few updates. We will show that the LP problem of [1] can be mapped onto piecewise linear models, which can be solved with the piecewise linear simulator PLATO. As we shall see in the results section, the PL simulator setup can calculate entire trade–off curves in about the same amount of CPU time as it takes to solve one LP problem for one solution point.

Setup of this paper

In section 2 we will discuss the LP formulation of the gate sizing problem, summarizing [1] and [3] chapters 4 and 6. In section 3 we will introduce the piecewise linear simulator PLATO. In section 4 we will discuss the mapping of the gate sizing problem onto piecewise linear models suitable for PLATO. In section 5 we will present results for all MCNC 91 benchmark examples, comparing the [1] approach with the performance of PLATO. In section 6 we will discuss some numerical aspects of optimizing very large circuits. In section 7 we will give some conclusions and directions of future work.

2 LP formulation of gate sizing

Basic delay model of a gate

To describe the way we have modeled the gate sizing optimization problem for the PL simulator, we will introduce a simple gate level delay model, as introduced in [1]. This is a model which uses the worst case delay of a logic gate, both with respect to inputs as with respect to rise– or fall time, as delay value. Although this kind of model can give quite accurate delay estimations for large circuits, it is not very accurate for many small circuits. It is, however, important to realize that any convex delay model can be used with our gate sizing method. In [6] the important class of distributed RC models is proved convex.

We use the following symbols: τ: delay of a gate, C: capacitance value, c: constant, S: speed factor

We start with the widely used basic model:

$$\tau_{gate} = \tau_{int} + c \times C_{load} \quad (1)$$

$$C_{load} = C_{wire} + C_{tr} \quad (2)$$

where C_{wire} is the wire capacitance and C_{tr} is the sum of the capacitances of the connected gates. Because we are dealing with sizeable gates, we introduce the *speed factor* S_{gate} of a gate:

$$\tau_{gate} = \tau_{int} + \frac{c \times C_{load}}{S_{gate}} \quad (3)$$

Because S_{gate} is implemented in the physical layout by multiplying the width of the transistors, the gate capacitances grow linearly with it, and we get:

$$C_{tr} = \sum_{i \in fanout(gate)} S_i C_{in,i} \quad (4)$$

Combining (2), (3) and (4) leads to a complete (nonlinear) delay model:

$$\tau_{gate} = \tau_{int} + c \times \frac{C_{wire} + \sum_i S_i C_{in,i}}{S_{gate}} \quad (5)$$

To use this delay model in a LP environment, it has to be linearized. To obtain the desired accuracy, we can use a piecewise linear fit with as many pieces as desired. In [3] it is shown, that 3 pieces is enough for above delay model, if we limit S to the range $[1, 3]$. Figure 1 shows a typical case of a 2–input nand gate in a 1.5µ CMOS process under various load conditions.

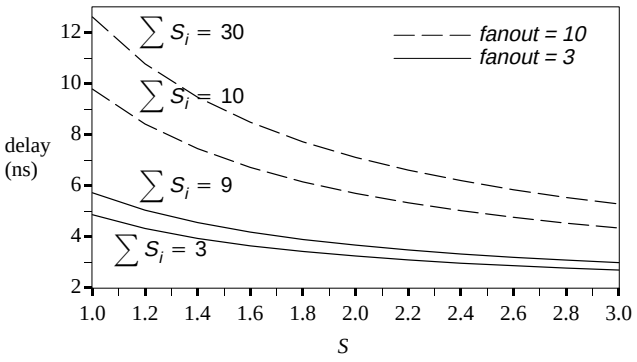


Figure 1 delay of a nand gate under different loads versus its speed factor

Delay of a logic circuit

We model the total delay of a logic circuit by calculating the longest path in the Boolean network [4]. To be able to do this, we introduce the symbol T_{gate}, the *schedule time* of a gate. It is defined as the worst case time after which the output of the gate will become stable on an input transition. The arrival times of the primary input signals are assumed to be known. The schedule time of a gate can be expressed in the schedule times of its input signals and its own delay:

$$T_{gate} = \tau_{gate} + \max_{i \in inputs(gate)} T_i \quad (6)$$

Both the summation and the max function can easily be expressed in LP terms.

The wire capacitance C_{wire} in (2) and (5) is estimated based on statistical data from actual layout. See [3] for details.

The total active area of a circuit

Because the transistor sizes in the gates are adapted in just one dimension with changing speed factor, the total active area of a circuit is a linear combination of the speed factors, where the constants c_i reflect the relative contribution to the active area of gate i.

$$A = \sum_{i \in gates} c_i S_i \quad (7)$$

CMOS dynamic power consumption

If we define $\bar{f}_{sw}$ to be the average switching frequency of a gate, the dynamic power consumption of a single CMOS gate can be expressed as:

$$\begin{aligned} P_{gate} &= \bar{f}_{sw} \times \frac{V_{dd}^2}{2} \times C_{load} \\ &= c_p \times C_{load} \\ &= c_p \times (C_{wire} + C_{tr}) \\ &= c_p \times \left(C_{wire} + \sum_{i \in fanout} S_i \times C_{in,i} \right) \end{aligned}$$

The total power consumption of a circuit is the sum of the power consumptions of the individual gates:

$$\begin{aligned} P &= \sum_{i \in gates} P_i \\ &= \sum_{i \in gates} c_{p,i} \times C_{wire,i} + \sum_{i \in gates} c_{p,i} \sum_{j \in fanout} S_j \times C_{in,j} \end{aligned} \quad (8)$$

The first summation of (8) is constant for our optimization problem, the second summation is linear in the speed factors.

To obtain the average switching frequencies, one must either perform logic simulations with appropriate input vectors or apply statistical methods, as in [7] and [13].

LP formulation of the gate sizing problem

The linear program is now composed as follows: Firstly, we define T_i to be the schedule time, τ_i the delay and S_i the speed constant of gate i. For the primary inputs T is the arrival time of the signal. The total delay of the circuit is:

$$T_{max} = \max_{i \in primary\ outputs} T_i \quad (9)$$

Now, for every gate in the circuit the following constraints are defined:

1. The n linearized delay models (a PL implementation of (5)):

$$\begin{aligned} &\tau_{gate} \geq c_{1,1} - c_{1,2} S_{gate} + c_{1,3} \sum_i S_i C_{in,i} \\ &\ldots, \\ &\tau_{gate} \geq c_{n,1} - c_{n,2} S_{gate} + c_{n,3} \sum_i S_i C_{in,i} \end{aligned} \quad (10)$$

2. S is limited:

$$S_{min} \leq S_{gate} \leq S_{max} \quad (11)$$

3. Definitions of schedule times (implement (6)):

$$\forall_{j \in fanin(gate)} T_{gate} \geq T_j + \tau_{gate} \quad (12)$$

4. Definitions for maximum schedule time of circuit (implement (9)):
   If the gate is a primary output:

$$T_{max} \geq T_{gate} \quad (13)$$

5. The objective function is a linear combination of A, P and T_{max}:

$$c_A A + c_P P + c_T T_{max} \quad (14)$$

If the circuit has a Boolean network representation with V vertices (one for every gate) and E edges (one for every connection), we can calculate the LP problem size. The number of constraints is $|E| + (2 + n)|V| + 4$ (1 for every predecessor relation, 2 per vertex for limitation of S_{gate}, n per vertex for the piecewise linear delay model, 1 to express the total delay T_{max}, 1 to express the total active area A, 1 to express the total power consumption P and 1 more to limit T_{max}). The number of variables is $3|V| + 3$ (per vertex S, τ, and T, and for the global network T_{max}, A and P).

Because in practical Boolean networks both the fanin and the fanout of a gate are limited, implying $|E| < c|V|$ for some constant $c > 1$, the size of the LP problem is effectively linear in the number of gates in the circuit.

3 The PL simulator PLATO

The simulator PLATO [5] is a piecewise linear simulator, primarily intended for simulating electrical and logical circuits. The component relations are described by a matrix, relating linear, dynamic and complementary variables and equations. The complementary variables and equations together form a Linear Complementarity Problem (LCP), which is the following problem: given a matrix M and a vector $\boldsymbol{q}$, determine vectors $\boldsymbol{w}$ and $\boldsymbol{z}$ satisfying

$$\begin{cases} \boldsymbol{w} = M\boldsymbol{z} + \boldsymbol{q} \\ \boldsymbol{w} \geq 0 \ , \ \boldsymbol{z} \geq 0 \ , \ \boldsymbol{w}^T\boldsymbol{z} = 0 \end{cases} \tag{15}$$

The equation $\boldsymbol{w} \geq 0$ is considered componentwise, i.e. $\forall_i \ w_i \geq 0$. The complementary variables and equations model the piecewise linear behavior of the components. Due to this piecewise linear modeling, electrical, logical and macro models can be used in one circuit description and simulation run. To support these different models, two types of connections (nets) are available: electrical, with voltage and current variables satisfying the Kirchhoff relations, and signal, which have only a voltage–like variable. The connections of a component to the nets, called terminals, also have one of these types. All kinds of components can be used by the simulator: it has no built–in models but uses a mixture of user–supplied models and library models. To solve the system efficiently, the following methods are repeatedly employed:

- The linear equations, determining the values of voltages, currents and signals, are separated from the component description. These equations are solved with an LU decomposition. Because of the sparse nature of the linear equations (connection matrix), a sparse data structure is used. If the linear equations change, the update of the LU matrices is calculated efficiently with an algorithm that visits only those elements of the matrices that change.
- The LCP is solved by a path–following algorithm devised by Van de Panne [14]. This algorithm follows the same path as the well–known Lemke algorithm, but only employs complementary (block–diagonal) pivots. The internal data structure is also block–diagonal, which is the reason why the performance of the Van de Panne algorithm is much better than the Lemke algorithm in PLATO. During the algorithm, the linear equations may change.
- The dynamic equations, a set of linear differential equations, are solved with an integration method. Which method is employed, depends on the problem; usually an implicit linear multi–step method is used. To exploit the sparsity and latency of most circuits, the integration method is employed in a multi–rate scheme, where the circuit is divided dynamically in clusters, and the components in one cluster have the same integration step size, differing from the step size in other clusters. However, if the solution is linear in time, the much simpler and more efficient Forward Euler integration method is used.

The simulator follows a path in the complex space of linear, dynamic and LCP variables. The starting point is determined first by applying the Van de Panne algorithm. The LCP variables and equations determine (convex) regions in the space of linear and dynamic variables. So the state of the LCP, the zero–nonzero partition of the $\boldsymbol{w}$ and $\boldsymbol{z}$ vectors, remains valid for some time during the integration. If an entry in either vector becomes negative during the integration, the Van de Panne algorithm is started to change the state of the LCP. The path that the simulator follows can be pictured as a piecewise continuous path (in time) through the space, mixed with (possibly discontinuous) steps. The continuous path is driven by the integration in time, while the (discontinuous) steps are governed by the Van de Panne algorithm.

4 Modeling in the PL simulator

To obtain the Area–Delay trade–off curve, the LP problem will be solved for a constraint $T_{max} \leq f(t)$ with $f(t)$ continuous. The function $f(t)$ may be non–monotone, but it is simpler to choose a function $f(t) = a - bt$, with $b > 0$ and a larger than $T_{max}(0)$, the value T_{max} assumes for $\sum S$ minimal (the value $T_{max}(0)$ can be determined easily from inspection of the circuit). Then the solution at $t = 0$ is feasible, and a list of solutions for different values of T_{max} is generated, until no solution can be found. This dynamic problem cannot be integrated easily into the used LP solver. However, the problem can be transformed into an LCP, as will be shown in the next paragraph. This LCP, together with the dynamic behavior, is solved by the simulator PLATO.

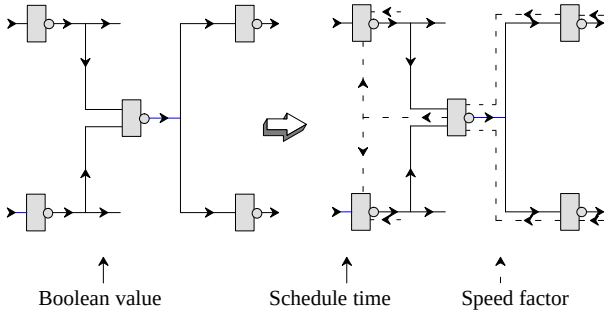
An LP problem is converted into an LCP in the following way. Let the LP problem be: find $\min_{A\boldsymbol{z}_1 \leq \boldsymbol{b}} \{ \ \boldsymbol{p}^T\boldsymbol{z}_1 \ \}$.

Apply the (Karush–)Kuhn–Tucker relations to find the system:

$$\begin{cases} \begin{pmatrix} \boldsymbol{w}_1 \\ \boldsymbol{w}_2 \end{pmatrix} = \begin{bmatrix} 0 & A^T \\ -A & 0 \end{bmatrix} \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix} + \begin{pmatrix} \boldsymbol{p} \\ \boldsymbol{b} \end{pmatrix} \\ \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix} \geq 0, \ \begin{pmatrix} \boldsymbol{w}_1 \\ \boldsymbol{w}_2 \end{pmatrix} \geq 0, \ \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix}^T \begin{pmatrix} \boldsymbol{w}_1 \\ \boldsymbol{w}_2 \end{pmatrix} = 0 \end{cases} \tag{16}$$

If a solution to this problem exists, the vector $\boldsymbol{z}_1$ is the solution of the original LP. For an LCP with a matrix as given in equations (16), the Van de Panne algorithm will always find a solution if it exists. This problem could be entered in a straightforward way into the circuit simulator by creating one large component with this system. But the problem can also be represented as the original circuit, with some additional special components. In this way, the sparsity of the network can be employed and the simulation will take much less computer time and resources. In the next section, the conversion of the LP as given in the equations (10) – (14) into a circuit with appropriate components is discussed.

Converting an LP into an LCP circuit



(a) Boolean network (b) circuit for optimization in PLATO

Figure 2 "Circuit" to perform optimization in the simulator

The LP problem of equations (10) – (14) is based on the subdivision of the logic circuit into connected components. To use the sparsity of the interconnections, the related LCP problem is converted into

a network. Instead of logic values, the schedule times and speed factors are passed between the components. As will be shown in the next paragraphs, it is simpler to use electrical nets: not only schedule times or speed factors are passed over a net, but also extra information needed to solve the optimization problem. Figure 2 shows the basic conversion step.

Each component has two output terminals, one with its schedule time as signal/voltage value, the other with its speed factor. Furthermore, the schedule times of the components in the fanin set of a component must be known, so the respective nets each have their own input terminal. For the same reason, the nets related to the speed factors of the components in the fanout set are connected to input terminals. The matrix of one component is constructed from three submatrices, each related to one group of inequalities.

The inequalities $S_{min} \le S_i \le S_{max}$ (equation (11)), together with the optimization requirement $\sum c_i S_i$ minimal (equations (7) and (14)), are transformed, according to (16), into the following equations:

$$\begin{cases} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} + \begin{pmatrix} c_i \\ S_{max} - S_{min} \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \\ S_i = z_1 + S_{min} \end{cases} \tag{17}$$

The equations $z_1 \ge 0$ and $w_2 \ge 0$ determine the minimum and maximum bounds of S_i, while the value c_i in the constant vector denotes the value in the objective function.

The second group of inequalities,
$T_i = \max\{ T_j \mid j \in fanin(i) \} + \tau_i$ (equation (12)), is transformed into system (18). For simplicity, the example gate has only two inputs with delays T_1 and T_2.

$$\begin{cases} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ z_1 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = \begin{bmatrix} w_{\alpha_1} \\ w_{\alpha_2} \\ w_1 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} \\ T_i = z_3 + \tau_i \end{cases} \tag{18}$$

The rows in the matrix related to the variables w_{α_1}, w_{α_2} and w_1 must be added to their respective rows. However, the problem with this set of equations is that the rows related with w_{α_1} and w_{α_2} are not in this component. These rows are the w_3 rows of their respective components. Therefore the value of z_4 respectively z_5 must be transported to these components and added to the relevant row. This is done conveniently by setting the schedule time explicitly to an electrical type, with as voltage value the schedule time T_1, and as current value on this connection the value of z_4. Because the currents satisfy the Kirchhoff law, the current on the output T is the sum of these currents of its successors. By adding one entry in the matrix in the row related with w_3, having the value -1 (incoming current!), the correct system is created.

The third group of inequalities, $\tau_i = \mathrm{PL}(S_i, S_{out})$ (equation (10)), has a form comparable to system (18). Therefore, the nets transferring the speed factors also have a current related to them, and an extra term occurs in the respective w_1 rows. The extra inequality $T_{max} < f(t)$ will give the same matrix as in equations (17), but with a right hand side $(0,\ f(t))^T$. This matrix is embedded in a final component that determines the maximal schedule time in the circuit by taking the maximum of the schedule times at the output gates.

The components in the circuit are created according to the equations above, and the connections between them are laid out according to the connections of the original circuit. The extra component for determining and changing the maximum schedule time is connected to the output gates of the circuit. The total number of LCP variables is $2((4 + n)|V| + |E| + 2)$, the number of linear (circuit) variables is $4|V| + |E|$. Notice that there is no special component connected to all gates for guiding the optimization process. The gates "know" themselves when their speed factors must change, as is explained in the next section.

Finding the LCP solution

The LCP solution is found by combining the Van de Panne algorithm with an integration in time. By using the inequality $T_{max} \le a - bt$, the relation between $\sum S$ and the time t is found. Then it is trivial to find the relation between T_{max} and $\sum S$. Using a linear function for the time, the numerical integration is as exact as possible and will give no problems. But first an initial solution must be found. This is rather straightforward, when starting from the basic point $z = 0$. This implies $S_i = S_{min}$ for all i, and the internal delays τ_i have the values related to these S_i. So only the schedule times T_i and the corresponding LCP variables must be determined. This is done easily by the Van de Panne algorithm. If the schedule time minus the delay of a component is less than the schedule time of one of its predecessors, the corresponding w_k is negative. By performing a block pivot, and due to the form of the equations, the corresponding w_k and z_k both become zero. In network terms, the schedule time is set to the schedule time of the predecessor plus the internal delay. If the initial solution is found, and the forced schedule time at this time point, a, is larger than T_{max}, all currents are zero at the starting point. The other case, that a faster initial solution is sought, will follow from the discussion in the next paragraphs. Because it is a feed–forward network, the calculations are easy and the T_{max} with $\sum S$ minimal is found.

The dynamic solution starts from this initial solution. With increasing time, the schedule time decreases until no faster solution can be found. The Van de Panne algorithm fails and the simulation stops with an error message. The dynamic solution will follow the piecewise linear trade–off curve exactly, due to the linear form of the constraint on T_{max}. We will describe the first step of the simulation process, and explain which actions will in general happen in the circuit in the subsequent steps. The first step starts at that time point of the simulation where the inequality in the last component, bounding the schedule time, just becomes invalid, and a new state of the LCP must be found.

The rest of this section is intended to give the reader some insight on the course of the optimization process. It is not mathematically rigorous, but it should give some feeling how we have constructed an "auto–optimizing" circuit.

The first step

Let k be the index of the violated inequality, i.e.

$$w_k = 0 \wedge \frac{\partial w_k}{\partial t} < 0.$$

Then the z_k variable can be increased. On circuit level, this means that a current will flow through a schedule time connection, towards the predecessor with the highest schedule time. In this predecessor, two actions happen. First, the current forces the schedule time of this cell to decrease. As in the previous cell, this creates a current through one schedule time connection towards a predecessor. So each cell on the critical path is activated by a current to decrease its schedule time. By the same process, the value of w_1 in each of these cells decreases. As soon as in one cell this value

reaches zero, the corresponding z_1 may increase. This means that the speed factor of that cell will increase. This decreases the delay of this cell, but increases the delay in its predecessors. However, the total schedule time over this path will decrease, because the global effect is used, i.e. if it would not decrease, w_1 would not decrease. So a new state of the LCP is found that will be valid for the next part of the simulation. The newly found state is translated into the linear equations. These equations can be interpreted by stating that the speed factor in this cell now depends on the time, and that the total schedule time only depends on the time by this relation.

Later steps

This process is repeated at each time point where the current state becomes invalid. This will happen if a speed factor reaches its maximum, if another piece of the piecewise linear approximation is reached, or if another path also becomes critical. In the first case, this cell is replaced by another cell (if it exists) and the curve can be tracked further. In the latter two cases, in general it is necessary to increase not only this speed factor, but also another cell's speed factor. Depending on the size of and the freedom in the circuit, in the latest stages of the simulation many speed factors are manipulated and a large part of all schedule times is changing. Instead of one gate on one critical path, many gates in a critical subnetwork are continuously considered.

Up till now, we have suggested that the speed factors will always increase. However, it may happen (see figure 5) that by decreasing one speed factor and increasing another at the same time the total schedule time will decrease. The speed factor may increase again later in the simulation. It is clear, that the schedule time of cells not on the critical path(s) may and will increase, whenever some of their successors become larger and faster.

5 Results

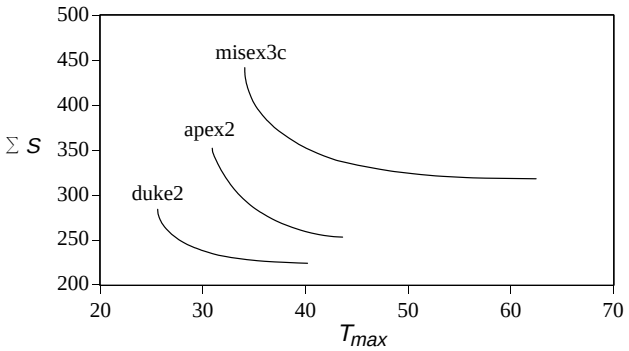


Figure 3 Area–Time trade–off curves for the circuits apex2, duke2 and misex3c, found by PLATO

The results can be divided in three parts:

1. the $T_{max} - \sum S$ trade–off curve, the actual result of the simulation run,
2. the comparison with the original LP solver, to compare the results, and
3. the different signal curves, describing the values of voltages and currents as functions of time. These can give detailed information on the optimization process, and can, for example, reveal delay bottlenecks in the circuit.

We have applied both the LP approach from [1] and the new PL simulator approach to the entire set of two–level examples of the MCNC benchmark suite [19], and to a group of other circuits. These other circuits are parameterized versions of a circuit that checks if a given *n*–bit number is prime, and, if not so, returns the smallest divisor. Example pr*n* is the *n*–bit version of this circuit. These prime number circuits have proved to be difficult examples for synthesis and layout software. All circuits were processed by the EUCLID logic synthesis system [2] until a netlist of basic gates was obtained. During this processing, the examples Z5xp1 and Z9sym became equal to 5xp1 and 9sym respectively. Therefore, Z5xp1 and Z9sym are not listed in the results. All experiments were performed on a HP 9000/750 workstation, running approximately 22 MFLOPS. The speed factor was limited between 1 and 3, and the PL approximation of the $\tau - S$ function had 3 pieces. The constants c_i in (7) and (17) were set to 1.

The $T_{max} - \sum S$ trade–off curve

The computation of this curve is the main result of this paper. The curves of three benchmark circuits are given in Figure 3. Other circuits give the same type of trade–off curves, whose particular shape will depend on the structure of the circuit. The circuits chosen for this figure are so complex that the trade–off curves seem smooth, but they are still piecewise linear. For simpler circuits, the curves have fewer sections. For more complex circuits, finding the complete trade–off curve is sometimes difficult, because of numerical problems in finding the left–most part of the curves, i.e. the fastest solutions with minimal T_{max}. These numerical problems are discussed in a later section.

Comparison with the LP solver

The solution of the dynamic LCP has to be compared with the LP solver with respect to the values of the solution, the run times and convergence properties. The results are presented in table 1. For both methods, the fastest solution found is given with its $\sum S$. For the LCP method, the run time for determining the complete trade–off curve is given, for the LP method only the run time for determining the fastest solution found is given. Furthermore, the number of gates and the number of linear and LCP variables in the simulator is given. The last two columns show the gain in speed and cost in extra area between the slowest (minimal $\sum S$) and fastest solution, determined from the results of the LCP. A %faster value of 55% means that the circuit after gate sizing has a delay of (100–55) = 45% of the original. No circuit can become more than 66.7% faster when S_{max} is 3. The values in inverted colors indicate solutions where both methods agree about the fastest solution.

Several aspects must be noted with respect to the results. The problem of finding the fastest solution becomes numerically less stable the larger the circuits are. Therefore, the LP solver could not always find this solution. In those cases the fastest solution which did converge is tabulated in table 1. For a few cases, the solution with minimal $\sum S$ could not be found with the LP solver, because the right–most part of the trade–off curve was too flat. This can probably be repaired by a more careful choice of constants in the objective function of the minimization problem. The simulator has comparable problems with the same circuits. For most circuits, the curve could be traced further (sometimes until the end) by changing the values of some internal numerical control parameters.

Table 1 indicates that both methods agree for most circuits on the results. Especially for the smaller circuits there is no doubt that both methods are equivalent, and only differ by small numerical errors. For the larger examples, it is not always clear which method is better. Because in PLATO the control of numerical errors is more carefully designed than in the LP solver, we suppose that PLATO gives slightly more accurate results.

The run times of both programs are comparable, although the LP solver calculates only one point of the trade–off curve. In figure 4 the run times are plotted against the size of the problem (in gates).

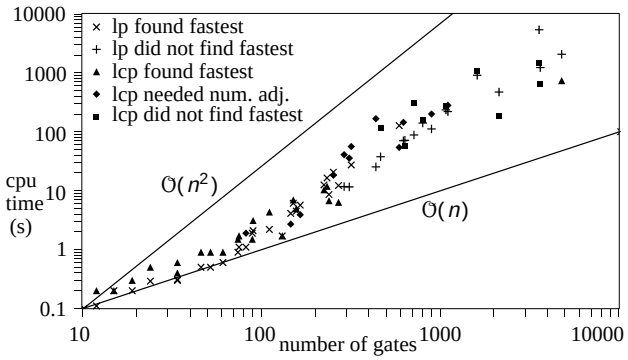


Figure 4 Run times versus number of gates

This figure suggests that the run times are $\mathcal{O}(n^2)$ for this range of problems. For the LP solver this is made plausible in [3]. The estimation of the order of the run times of the simulator is complex, because all calculations are performed on sparse data structures, so the density of the linear equations and the connectivity in the system, and the number of time steps determine the run time. If the connectivity and density remain bounded, the run time is linear in the number of time steps, and is expected to be $\mathcal{O}(n)$. However, in the later part of the simulation, during the determination of the fastest possible solution, the connectivity and density grow (as will be explained in the next section). This might explain why the run time tends to grow as $\mathcal{O}(n^2)$.

The cost and gain of the fastest solution compared with the initial, slowest solution show a wide range of values. The schedule time may vary between 16% to 66% faster, while the extra cost in area may differ from 0.4% up to 172%. There is a tendency for small circuits to have low gain at relatively high costs, while for large circuits high gains are obtained at low costs. This can be explained by the fact that in small circuits there is not much freedom, because each path in the Boolean network contains only a few gates. After a few steps of the simulation, most of the network becomes critical, so decreasing the schedule time is only possible by increasing many speed factors simultaneously. Larger circuits usually have one or two long paths, so by increasing only the speed factors on these paths a faster solution can be found.

Signal curves

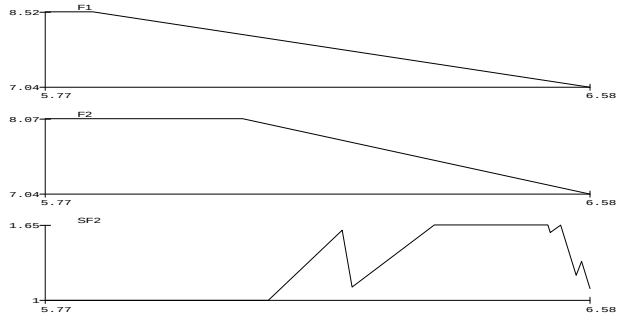


Figure 5 Schedule times of outputs F1 and F2 of circuit con1, and speed factor of gate F1

For each circuit all values as function of the simulation time can be printed. We have chosen to show only three values of the simplest circuit in the benchmark suite, con1 (Figure 5). The reason is that many values are trivial (the component's speed factor remains 1) or uninteresting (the "currents" in the circuit). A third reason is that most values show the behavior that is expected from the model, i.e. the speed factors increase monotonously, and the schedule times show a mixture of increasing and decreasing values, depending on its place in the network. One of the most interesting features, which is many times ignored, is that the speed factors may decrease during the simulation, because it gives room for other speed factors to increase and so decrease the total schedule time of the circuit. This behavior is shown exactly in Figure 5, where the schedule times of the two outputs of the circuit are shown. First gate F1 is on the critical path, later both gates are on it. The speed factor of gate F2 shows an irregular behavior during the simulation. This can be explained by the fact that this gate has a low internal delay, so decreasing this delay is not so interesting. At certain values for the forcing schedule time, it is therefore advantageous to decrease the speed factor. Figure 5 shows that for the fastest solution only a small speed factor is found, while for some slower solutions a larger speed factor suits better. Many heuristical approaches to transistor sizing ignore the fact that sizes must sometimes decrease during the optimization process to stay near the optimal solution.

6 Numerical aspects

One of the important conclusions that can be drawn from the experiments is that both solution methods for this type of problems show numerical problems for large examples. We will try to analyze these problems in qualitative terms. The characteristics for the problems that the methods can not solve are the same, i.e. both the LP solver and PLATO do not find the fastest solution for (most) problems exceeding a size of about 1000 gates. The symptoms are in many cases the same, namely that a pivot can not be performed because it is too small. Furthermore, the run time increases disproportionate when finding a faster solution, so most time is spent in the left–most part of the trade–off curve.

This last symptom can be explained by the fact that (in the simulator) the length of the time interval between two subsequent changes of the state of the LCP shortens and the number of matrix entries increases. These facts can be explained by the increasing interdependency between the gates, so more speed factors are changed to decrease the total schedule time.

There may be two reasons for the numerical problems: the matrix of the LP/LCP problem may be ill–conditioned, and/or the convex hull spanned by the inequalities is very flat near the optimal solution. The first case is not likely, as can be seen from the inequalities (10) – (13). The coefficients are all $\mathcal{O}(1)$, so an ill–conditioned matrix will occur if two inequalities determine nearly equal hyperplanes. This is not the case. For the LCP the same reasoning shows that both the linear equations and the LCP equations are not ill–conditioned. So it is most likely that the convex hull spanned by the inequalities is very flat near the optimal solution.

7 Conclusions

The approach to compute trade–off curves for gate sizing with a piecewise linear simulator has proved to be very effective. Entire trade–off curves can be computed with about as much CPU time as it takes to get one point on the curve with the PL approach. There are some numerical problems with circuits with more than 1000 gates, but it should be noted that for these large circuits still a substantial part of the trade–off curve is obtained. Because this is from the designers point of view probably the most interesting part (the right part), these partial results are still useful.

The topic of numerical stability will be subject of further research, because we believe that by careful analysis of the source of the numerical problems we might find a way to avoid them.

Table 1: Results, run times, characteristic sizes and gain / cost for solving LP and LCP problem

Name	LP	LCP	# gates	# linear vars	# LCP vars	% faster	% more S
	run (s)	run (s)					
5xp1 [1]	4.1	2.7	146	721	1285	**29.6**	34.1
9sym [1]	1.1	1.9	82	400	830	**17.6**	50.2
alu4 [1]	126.7	54.2	589	3722	287	**56.9**	14.6
apex1 [1]	220.6	281.0	1103	7639	11126	**44.0**	11.7
apex2	20.5	18.3	253	1488	2474	**29.6**	39.1
apex3 [2,3,4]	470.8	182.9	2131	10722	20934	64.5	0.4
apex4 [2,3,4]	1229.3	639.9	3622	18551	35911	66.5	1.1
apex5 [1,2]	25.2	166.4	437	2685	4289	44.3	39.3
b12	0.5	0.9	46	250	444	**22.0**	53.4
bw	1.9	1.5	89	522	934	**26.2**	25.6
clip	2.1	3.1	90	554	915	**26.9**	45.7
con1	0.1	0.2	12	62	118	**17.4**	127.6
cordic	0.6	0.9	61	316	565	**18.8**	72.0
cps [1,2]	70.3	144.6	622	4281	6171	**53.9**	20.6
duke2	12.5	10.3	224	1363	2209	**36.4**	26.8
e64	8.6	6.7	239	1330	2174	**31.2**	22.9
ex1010 [2,3]	234.9	273.7	1076	7203	10424	38.7	8.9
ex4	12.2	6.3	270	1477	2669	**28.5**	25.4
ex5	16.4	11.8	233	1545	2280	**41.4**	19.2
inc	0.5	0.9	52	312	543	**21.8**	34.2
misex1	0.3	0.6	34	177	336	**22.2**	62.1
misex2 [1]	5.7	3.9	165	940	1543	**27.1**	28.2
misex3 [2,4]	2063.1	736.5	4769	21818	47470	66.7	0.8
misex3c [1]	27.2	56.1	318	1991	3285	**46.0**	38.9
o64	4.5	4.9	157	661	1388	**22.6**	172.0
pdc [1,2]	11.6	35.6	310	1969	3009	**38.6**	28.3
rd53	0.2	0.3	19	88	210	**13.8**	47.4
rd73	1.1	1.7	75	452	760	**21.3**	31.6
rd84	1.7	1.7	131	595	1334	**19.1**	14.3
sao2	0.9	1.5	74	356	758	**18.8**	34.3
seq [1,2]	111.3	200.6	894	6021	8878	56.9	16.9
spla [2]	11.7	40.8	290	1870	2844	**35.0**	29.0
t481	0.3	0.4	34	181	315	**18.0**	75.3
vg2	2.2	4.3	111	638	1093	**28.6**	44.3
xor5	0.2	0.2	15	95	168	**16.5**	77.8
pr8	6.1	6.9	151	953	1590	**37.3**	26.5
pr9 [2,3]	37.6	117.3	464	2849	4377	45.3	36.0
pr10 [2,3]	140.0	156.0	799	5754	8190	47.9	15.3
pr11 [2,3]	910.0	1059.0	1609	11750	16695	52.8	6.5
pr12 [2,3]	5380.0	1438.8	3569	26325	37166	49.2	2.2

[1] LCP found fastest solution by adjusting numerical control parameters

[2] LP could not find fastest solution

[3] LCP could not find fastest solution

[4] LP could not find slowest solution

Literature

[1] BERKELAAR, M.R.C.M. and J.A.G. JESS, "Gate Sizing in MOS Digital Circuits with Linear Programming", *Proceedings of the European Design Automation Conference 1990*, pp. 217–221.

[2] BERKELAAR, M.R.C.M. and J.F.M. THEEUWEN, "Real Area–Power–Delay Trade–off in the EUCLID Logic Synthesis System", *Proceedings of the IEEE Custom Integrated Circuits Conference 1990*, pp. 14.3.1–14.3.4.

[3] BERKELAAR, M.R.C.M., "*Area–Power–Delay Trade–off in Logic Synthesis*", Ph.D. Thesis Eindhoven University of Technology, Eindhoven, The Netherlands, 1992.

[4] BRAYTON, R.K., R. RUDELL, A.L. SANGIOVANNI–VINCENTELLI and A. WANG, "MIS: A Multiple–Level Logic Optimization System", *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, Nov. 1987, Vol. CAD–6, pp. 1062–1081.

[5] BUURMAN, H.W., "*From Circuit to Signal: development of a piecewise linear simulator*", Ph.D. Thesis Eindhoven University of Technology, Eindhoven, The Netherlands, 1993.

[6] FISHBURN, J.P. and A.E. DUNLOP, "TILOS: A Posynomial Programming Approach to Transistor Sizing", *Proceedings of the IEEE International Conference on Computer–Aided Design 1985*, pp. 326–328.

[7] GHOSH, A., S. DEVADAS, K. KEUTZER and J. WHITE, ''Estimation of Average Switching Activity in Combinational and Sequential Circuits", *Proceedings of the 29th ACM/IEEE Design Automation Conference 1992*, pp 253–259.

[8] GLASSER, L.A. and L.P.J. HOYTE, "Delay and Power Optimization in VLSI Circuits", *Proceedings of the IEEE Design Automation Conference 1984*, pp. 529–535.

[9] HEDLUND, K.S., "Models and Algorithms for Transistor Sizing in MOS Circuits", *Proceedings of the IEEE International Conference on Computer Aided Design 1984*, pp. 12–14.

[10] KAO, W.H., "Algorithms for Automatic Transistor Sizing in CMOS Digital Circuits", *Proceedings of the 22nd Design Automation Conference 1985*, pp. 781–784.

[11] MARPLE, D., "Transistor Size Optimization in the Tailor Layout System", *Proceedings of the IEEE Design Automation Conference 1989*, pp. 43–48.

[12] MATSON, M.D., "Optimization of Digital MOS VLSI Circuits", *Proceedings of the Chapel Hill Conference on VLSI 1985*, pp. 109–126.

[13] NAJM, F.N., ''Transition Density, a Stochastic Measure of Activity in Digital Circuits", *Proceedings of the 28th ACM/IEEE Design Automation Conference 1991*, pp 644–649.

[14] PANNE, C. VAN DE, "A Complementary Variant of Lemke's Method for the Linear Complementarity Problem", *Mathematical Programming*, 1974, Vol. 7, pp. 283–310.

[15] RUEHLI, A.U., P.K. WOLFF and G. GOERTZEL, "Power and Timing Optimization of Large Digital Systems", *Proceedings of the IEEE International Symposium on Circuits And Systems 1976*, pp. 402– 405.

[16] SAPATNEKAR, S.S., V.B. RAO and P.M. VAIDYA, "A Convex Optimization Approach to Transistor Sizing for CMOS Circuits", *Proceedings of the IEEE International Conference on Computer Aided Design 1991*, pp. 482–485.

[17] SAPATNEKAR, S.S., V.B. RAO, P.M. VAIDYA and S.M. KANG, ''An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Optimization", *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 11, November 1993, pp. 1621–1634.

[18] SHYU, J., A. SANGIOVANNI–VINCENTELLI, J.P. FISHBURN, and A.E. DUNLOP, "Optimization–Based Transistor Sizing", *IEEE Journal of Solid–State Circuits*, Vol. 23, No. 2, April 1988, pp. 400–409.

[19] Yang, S., "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0", *Report of the Microelectronics Center of North Carolina*, 1991.