# Glitch Power Minimization by Selective Gate Freezing

Luca Benini, *Member, IEEE*, Giovanni De Micheli, *Fellow, IEEE*, Alberto Macii, *Student Member, IEEE*, Enrico Macii, *Member, IEEE*, Massimo Poncino, *Member, IEEE*, and Riccardo Scarsi

***Abstract*—This paper presents a technique for glitch power minimization in combinational circuits. The total number of glitches is reduced by replacing some existing gates with functionally equivalent ones (called F-Gates) that can be "frozen" by asserting a control signal. A frozen gate cannot propagate glitches to its output. Algorithms for gate selection and clustering that maximize the percentage of filtered glitches and reduce the overhead for generating the control signals are introduced. A power-efficient CMOS implementation of F-Gates is also described. An important feature of the proposed method is that it can be applied *in place* directly to layout-level descriptions; therefore, it guarantees very predictable results and minimizes the impact of the transformation on circuit size and speed.**

***Index Terms*—CMOS digital integrated circuits, design automation, power optimization.**

## I. Introduction

SPURIOUS transitions (also called *glitches*) in combinational CMOS logic are a well-known source of unnecessary power dissipation [1]. Reducing glitch power is a highly desirable target because in the vast majority of digital CMOS circuits, only one signal transition per clock cycle is functionally meaningful. Unfortunately, glitch power is heavily dependent on the low-level implementation details, namely, gate propagation delays and input transitions misalignments. For this reason, glitch power estimation requires accurate simulation tools with precise gate and transistor delay models.

In this paper, we propose an automatic circuit transformation technique for glitch power reduction. Minimizing glitching at the gate level is a complex task because it is difficult to estimate the impact that circuit transformations can have on glitch power. Two different approaches have been taken to solve this problem. Networks can be designed using a glitch-free implementation style (such as Domino [2] or Shannon [3] circuits); alternatively, a nonglitch-free implementation can be optimized to reduce the number of glitches. The first solution has a major limitation: glitches are eliminated, but the constraints imposed by the design style may lead to circuits that are less power efficient than those realized with standard static CMOS gates. The second approach is hindered by the uncertainties in the estimation of the cost function that drives the optimization procedure. In other words, it is difficult to estimate if the changes in the network introduced to minimize glitching are useful or not because they may modify the delay distribution of the circuit in a quite unpredictable fashion.

We present an incremental optimization technique that reduces glitching in standard static CMOS implementations, but we overcome the predictability problem by posing tight constraints on the amount of network perturbation that can be introduced by the glitch minimization procedure. More in detail, we propose an optimization method that operates *in place* on a layout-level description. We perform minor modifications of the netlist that can be implemented on the placed and routed circuit only by applying partial rewiring of a few signal nets. The cost function that controls the optimization is very accurate because glitch and total power estimation are carried out on a network with back annotation of wiring loads.

The procedure for glitch minimization is based on a well-known idea. Glitches are eliminated by adding some redundant logic that prevents spurious transitions. This can be done by inserting latches in a gate-level netlist and controlling the *latch-enable* pins with redundant signals generated *ad hoc*. We propose an alternative technique, called *gate freezing*, that is much less perturbative of the existing circuit structure. Gates with high spurious activity are replaced with functionally equivalent ones (called *F-Gates*) that can be made insensitive to input transitions by asserting a control signal (called *C-Signal*). Although this is functionally equivalent to latch insertion, it can be implemented much more efficiently (area and power-wise) and the modified gates can replace directly the original gates in a placed and routed netlist.

The overhead of control-signal generation is reduced by selecting only a subset of the gates in the original implementation for replacement with *F-Gates* and by grouping the *F-Gates* in clusters that share a common *C-Signal*. We describe algorithms for the selection and clustering of candidate *F-Gates* that maximize power savings by maximally reducing glitches and by minimizing the additional power consumed by the control logic that generates the *C-Signals*. We also propose an efficient implementation style for the *F-Gates*, and we experimentally validate it through a number of low-level (i.e., Spice) simulations. Finally, we describe a low-power implementation of the *C-Signal* drivers.

We have applied the gate freezing procedure to the full suite of the ISCAS'85 benchmarks [4], as well as to some examples taken from the MCNC'91 suite [5]. We have achieved an average glitch power reduction of 14.0%, resulting in an average total power savings of 6.3%. Area is only marginally increased

Manuscript received February 2, 1999; revised September 22, 1999.

L. Benini is with the Dip. di Elettronica, Informatica e Sistemistica, Università di Bologna, Bologna 40136, Italy.

G. De Micheli is with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 USA.

A. Macii, E. Macii, M. Poncino, and R. Scarsi are with the Dip. di Automatica e Informatica, Politecnico di Torino, Torino 10129, Italy.

Publisher Item Identifier S 1063-8210(00)04350-X.

(2.8% in average), and speed is unchanged. Obviously, gate freezing only targets circuits with high glitching; therefore, its applicability is of limited interest in the cases where spurious transitions have a negligible impact on the global power budget.

The outline of the paper is as follows. Section II summarizes the existing previous work on glitch power minimization. Section III outlines the optimization paradigm based on gate freezing and provides details on how *F-Gates* and *C-Signals* can be efficiently realized. In Section IV, the algorithms and heuristics we adopted to implement the various steps of the gate freezing procedure are illustrated. Section V reports the experimental results concerning the application of gate freezing to a set of standard benchmarks, while Section VI collects Spice simulation data for some *F-Gate* cells. Finally, Section VII concludes the paper with some final remarks.

## II. Previous Work

The development of automatic techniques for glitch minimization in logic circuits has been the subject of intensive research in the past, since networks with limited spurious activity are usually highly desirable. In some cases, the elimination of glitches is required for correct circuit operation. For example, in [6] the target was the elimination of critical races in asynchronous circuits.

In this section, we briefly summarize a number of existing approaches to glitch power reduction; some of them have been developed for specific applications [7], [8], while others have general-purpose [9]–[11]. We also discuss a technique, known as guarded evaluation [12], that, although not explicitly targeting glitch power minimization, has some affinity with the gate freezing idea we present in this paper.

Special-purpose techniques have focused on arithmetic circuits. In [7], a self-timed method is employed to prevent the propagation of spurious transitions to the outputs of a carry-lookahead adder. Array multipliers are considered in [8]. The authors observe that multipliers are generally very glitchy and multiple spurious transitions are propagated within the array of the partial products. The propagation of such glitches is reduced by inserting *trasition-retaining barriers* (i.e., sets of latches) in the array. Latches are controlled with a self-timed signal that is generated by additional logic. The power dissipated by such logic is more than compensated by the power reduction obtained by eliminating the glitches.

General-purpose techniques have, in general, wider applicability; in addition, they are more suited for implementation within computer-aided design tools. A retiming transformation is proposed in [9]. First, the circuit is analyzed to detect gate outputs with multiple spurious transitions and high fanout. Then, retiming is applied with the objective of moving the flip-flops on the outputs of the target gates. Retiming moves are limited by timing and area constraints. The main shortcoming of this technique is that it is applied before placement and routing. Since retiming is a quite "intrusive" transformation that implies changes in the clock distribution, as well as in the number and position of the flip-flops, it is difficult to apply it to a circuit in an incremental fashion after placement and routing. As a consequence, the estimated savings may diminish or get canceled after the layout phase.

Gate resizing [11] has been applied to glitch power minimization. This technique eliminates glitches by equalizing all path delays in a combinational logic network. Path equalization is obtained by downsizing gates that do not lie on critical paths, thereby slowing down fast propagation paths, without changing the worst case delay of the circuit. Effective path equalization by resizing requires large technology libraries with many differently sized functionally equivalent logic gates. In addition, perfect equalization can be difficult to achieve if path delays are distributed very unevenly. In fact, if the delay of a short path is still smaller than that of the critical path when all gates on such a short path have been scaled down to minimum size, spurious activity remains possible (multipliers are typical examples of circuits with widely different path delays). Finally, the resizing approach presented in [11] is applied before physical design. Therefore, the cost metric employed for driving the resizing procedure has limited accuracy.

Guarded evaluation [12], even though not explicitly targeted toward glitch power minimization, exploits automatic latch insertion to reduce power consumption. In this approach, the internal signals that are available in the combinational network are used as latch-enable controls. The rationale in guarded evaluation is that of preventing the switching of nonobservable sections of a combinational logic network by inserting latches controlled by signals that imply nonobservability. The method has produced promising results, but it suffers from the same drawback of the retiming technique, i.e., it is applied before physical synthesis. Latch insertion significantly perturbs both cell count and netlist connectivity; thus, postlayout results are not very predictable and optimization may be quite inaccurate.

A number of register-transfer (RT) level transformations for glitch power optimization have been introduced in [10]. Here, the focus is on glitches generated by misaligned control signals that have long propagation paths within the data path and cause sizable power consumption. The main objection to this approach is that at the RT level only approximate timing information is available, and the glitch power reduction predicted at this level may be an inaccurate estimate of the result achieved after logic synthesis, tech mapping, placement, and routing. On the other hand, a key advantage of the method is that it performs local low-impact transformations on control signals that fan out to large portions of the data path. The authors show that RT-level transformations reduce glitching by a vast amount that is unlikely to be completely canceled in the later phases of the synthesis flow.

In the next section, we describe a methodology for glitch power minimization that works at a much later stage in the design process, i.e., it is applied to layout-level descriptions. Thus, it produces much more predictable savings. On the other hand, it might not be as effective as RT-level techniques, since it operates in a more constrained environment, where reduced degrees of freedom are left for optimization.

## III. Gate Freezing

The flow of the gate freezing procedure starts from a placed and routed combinational circuit (possibly contained in a larger

design). We assume that wire loads have been extracted and back annotated into the gate-level circuit model. Accurate switch-level simulation of user-specified patterns is carried out to obtain statistics of the glitching activity. Static gate-level timing analysis is also performed, and arrival and required times are computed for each node in the network. The information on glitching and timing constraints is subsequently exploited in the gate selection step.

A fraction of the gates in the netlist is selected. We choose noncritical gates with high glitching and high fanout (the selection process is described in detail in Section IV). The selected gates are then replaced with functionally equivalent *F-Gates* with one additional control input. Whenever the control signal is low, the gate is not sensitive to input transitions. Glitches on the output of the gate can thus be eliminated by first keeping the control signal low at the beginning of the clock cycle until the input signals of the gate have stabilized to their final value and then by raising it and keeping it high until the end of the clock cycle. The cost of the generation of the control signals is amortized by *clustering* the *F-Gates*. Gates in a cluster share a single control signal. The last step in the gate freezing procedure is the incremental modification of the layout. The selected gates are replaced by *F-Gates*, the drivers for the control signals are instantiated, and the control signals are routed.

A practical implementation of gate freezing, to be successful, must satisfy two critical requirements. First, the overhead for the generation of the control signals should be more than paid off by the power saved with gate freezing. Second, the physical implementation of the circuit should be minimally modified by the transformation. Hence, we need low-overhead implementations for *F-Gates* and *C-Signal* generation circuitry; moreover, we need effective algorithms for selecting target gates and for grouping them into clusters that share a common *C-Signal*. The next two sections are dedicated to the description of efficient implementations for *F-Gates* and *C-Signals*. The algorithms for gate selection and clustering are described in Section IV.

### A. F-Gates

Once a target cell in the tech-mapped circuit has been selected, it is replaced by a modified library cell (the *F-Gate*) whose output can be selectively "frozen" with the purpose of reducing the amount of glitching.

The basic modification of a generic CMOS library cell is shown in Fig. 1. It consists of the insertion of a n-type transistor in series with the n network. The gate input of this n-type transistor is driven by the *control input* $C$.

The behavior of the modified gate is quite intuitive: When the control input $C$ is high, the gate operates normally; on the other hand, when $C$ is low, the gate output is disconnected from the ground and, therefore, it can never be discharged to the logic value 0.

In this configuration, the output of the gate is only partially "frozen"; in fact, only the one–zero transition is actually forbidden, whereas the gate output can still exhibit the zero–one transition. This may occur for any input configuration that is supposed to force a one on the output. In other terms, a low control input $C$ will never allow a gate output that is at the logic value 1 to make a transition.

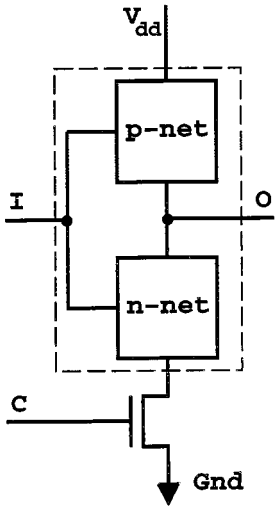



Fig. 1. Basic transformation of a library cell.

The discussion above implies that we do not guarantee to completely filter out transitions on the gate output. One trivial way to obtain a total filtering would be to mimic the structure of dynamic gates, and symmetrically insert a p-type transistor before the supply connection, in series with the p network; in this case, the gate terminal of this p transistor should be controlled by the other phase, $\bar{C}$, of the control input $C$.

This solution would suppress any transition on the gate output, and solve the limitation of the single n-transistor solution. However, there are some considerations that make the introduction of the p transistor undesirable.

- The load on the signal $C$ is doubled for any gate such signal must drive.
- Both phases of signal $C$ are required.
- The area of the modified gate is sensibly larger than in the case of the single n transistor; this is because, to guarantee the proper output signal levels, the p transistor must be larger than the n transistor.

The single n-transistor solution is less intrusive with respect to the size of the library cell and its speed; therefore, it is preferable to the complementary one. Nevertheless, the single n-transistor configuration will obviously be slightly larger and slower than the original cell.

An additional signal integrity issue must be mentioned. Consider the situation where signal $C$ is zero and the configuration at the gate's inputs is such that the output is expected to be zero. Since there is no path to ground, the gate output is actually in a floating state. If, for some reason, signal $C$ stays low for a relatively long time, the floating (high-impedance) value at the gate output may deteriorate because of parasitic coupling or junction leakage current. When the nodal voltage reaches an intermediate value between the power supply and ground, it could create a conducting path from supply to ground in fanout gates, causing static power consumption.

During normal operation, the situation mentioned above does not occur, since signal $C$ is guaranteed to be driven high for at least a fraction of each clock cycle. However, in some architectural schemes, where the clock is "controlled" by internal signals (e.g., gated-clock configurations [13]), the clock may

actually be shut down for a relatively long time. In these architectures, the control input of the *F-Gate* requires one extra condition, that is, $C$ must be one whenever the conditions for clock gating are satisfied.

### B. C-Signal Generation

As described at the beginning of this section, static timing analysis is carried out on the initial mapped circuit in a preprocessing phase. For each gate $g$, arrival time $\mathrm{AT}(g)$, required time $\mathrm{RT}(g)$, and slack $S(g)$ are available.

Once the nodes in the network have been ranked according to their suitability using the criteria described in Section IV-A, we must restrict the choice for the replacement with a modified cell only to gates that are not on the critical path (i.e., gates with nonnull slack). This is because, even though the modified cells are only slightly slower than the standard ones, replacing a critical gate will cause the cycle time of the circuit to increase.

We now derive the timing and functional conditions that define the behavior of the control signal $C$ for a candidate gate $g$. In the following, let $T$ denote the clock period, $g$ the candidate gate, and $\mathrm{AT}(g)$ its arrival time. Moreover, let $X = (x_1, \cdots, x_n)$ denote the inputs of $g$, and $\mathrm{AT}(x_i), \mathrm{RT}(x_i), i = 1, \cdots, n$ their corresponding arrival and required times.

We should observe first that all transitions occurring prior to the arrival time are glitches, and can thus be suppressed. This implies that $C$, the control signal of the modified gate that will replace $g$, can be held at zero in the time interval between the beginning of the clock cycle and the time $\Delta$ that equals the latest arrival time of the inputs of $g$. In symbols

$$\Delta(g) = \max_{i=1,\cdots,n} \mathrm{AT}(x_i). \tag{1}$$

At time $\Delta(g)$, gate $g$ will have all of its inputs "ready" and will be ready to propagate its final value; this implies that signal $C$ should go high at time $\Delta(g)$ in order to allow gate $g$ to exhibit a correct temporal behavior.

However, the control signal $C$ does not actually need to go high exactly at $\Delta(g)$. As a matter of fact, if $g$ is noncritical, all of its inputs will be noncritical as well. Therefore, all the inputs of $g$ will have some slack. What we must then guarantee is that $C$ goes to one (i.e., $g$ is free to evaluate) *before* the time $\Gamma$ which is the earliest *required* time of all inputs of $g$. In symbols

$$\Gamma(g) = \min_{i=1,\cdots,n} \mathrm{RT}(x_i).$$

This provides a safety margin to the transition of the control signal because we have a *don't care* region between $\Delta(g)$ and $\Gamma(g)$ where we can decide whether to raise signal $C$ or not.

In summary, signal $C$ should have a timing behavior as the one depicted in Fig. 2. $C$ can be thought of as a delayed copy of the clock, yet with a different duty cycle $D_C = 1 - \Delta(g)/T$. In the picture, the shaded area shows the slack for the zero–one transition on $C$. With such signal, we can guarantee that all the glitches before the latest arrival time of the gate's inputs are filtered out; the only spurious transition that can propagate through a gate is the zero–one transition mentioned in Section III-A. The shaded area in the figure shows this glitch-free portion of a clock cycle.

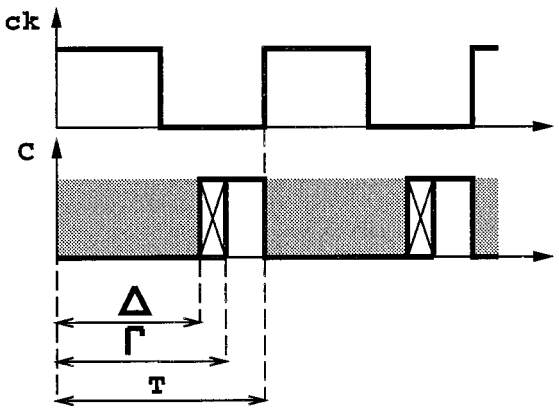



Fig. 2. Relation between signal $C$ and the clock.

Signal $C$ can be derived by proper filtering of the clock signal, ck. As mentioned above, in the design of the circuit for the generation of the control signals, we limit ourselves to considering the $\Delta$'s, and use the slack of the control signal $\Gamma - \Delta$ only as a safety margin.

We can derive the following relations between the clock signal ck and the control signal $C$. If the time $\Delta$ is smaller than the fraction of the clock period with $\mathrm{ck} = 1$, $C$ can be obtained as: $C = \mathrm{ck}' + \mathrm{ck}_\Delta$, where $\mathrm{ck}'$ is the inverse of the clock signal, and $\mathrm{ck}_\Delta$ is the clock signal delayed by an amount of $\Delta$. Conversely, if $\Delta$ is greater than the fraction of the clock period with $\mathrm{ck} = 1$, $C$ is computed as: $C = \mathrm{ck}' \cdot \mathrm{ck}_\Delta$. The above relations are shown by means of timing diagrams in Fig. 3(a) and (b), respectively.

Generating $C$ requires a delayed version, $\mathrm{ck}_\Delta$, of the clock signal. The trivial implementation of a delay element is a chain of an even number of suitably sized inverters. This solution is highly power and area demanding when the delay is much larger than that of a single inverter. More efficient implementations have thus been proposed [14], [15]. We adopted the one of [15] (see Fig. 4), whose power dissipation is approximately three times that of a single inverter and the area is around four times larger (when the delay is approximately 16 times that of a single inverter).

The $C$ signals are distributed through a dedicated network (made of tapered buffers and delay elements) that stems from the clock tree in a single point. Since the tap point is a minimum-size buffer, the additional load on the clock tree is usually negligible.

## IV. Gate Selection and Clustering

The basic version of the gate freezing algorithm is shown in Fig. 5. The procedure takes a placed and routed circuit $M$, a library $L$, containing both ordinary cells and the corresponding *F-Gates*, and the maximum number $N$ of gates to be selected for replacement with *F-Gates*. Static timing analysis (Line 1) and power estimation (Line 2) are first performed; then, candidate cells for replacement with *F-Gates* are selected (Line 3). Finally, each selected gate $g$ is replaced with the corresponding *F-Gate* (Line 4), the timing for the control signal of such a gate is computed (Line 5), and the circuitry for generating it is instantiated (Line 6). We first analyze procedure `SelectCandidates` (Section IV-A), which returns the set of gates to be

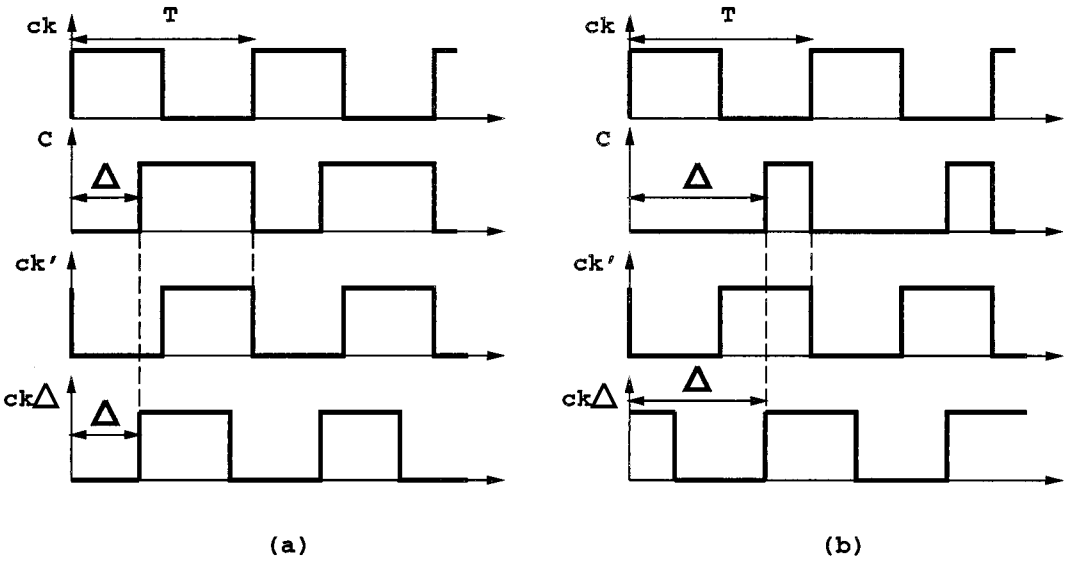



Fig. 3. Timing diagrams of control signal generation.

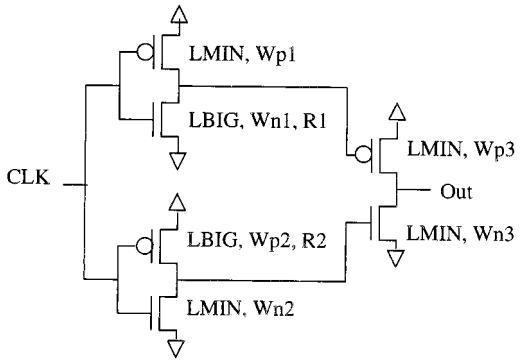



Fig. 4. Low-overhead implementation of a delay element.

```
procedure GateFreeze(M, L, N) {
1  (RT[ ], AT[ ]) = StaticTimingAnalysis(M);
2  Power[ ] = PowerSimulation(M);
3  G = SelectCandidates (M, Power, AT, N);
   foreach (gate g in G) {
4      M = ReplaceGate (M, L, g);
5      F_g = ComputeControlCirc (M, AT(g), T);
6      M = AddControlCirc (M, g, F_g);
   }
}
```

Fig. 5. Gate freezing algorithm.

replaced by *F-Gates*. Then, we focus our attention on gate clustering (Sections IV-B and C), a key step that is required to make gate freezing applicable in practice.

### A. Selection of the Target Cells

The selection of the cells to be replaced with *F-Gates* is mainly driven by the amount of glitching observed at the output of each cell. Additionally, we account for the capacitive load of the nodes. Therefore, our procedure for cell selection is to sort the network nodes in decreasing order of glitching activity weighted by load capacitance (we call this metric

```
procedure ClusteredGateFreeze(M, L, N) {
1  (RT[ ], AT[ ]) = StaticTimingAnalysis(M);
2  Power[ ] = PowerSimulation(M);
3  G = SelectCandidates(M, Power, AT, N);
4  (G, P) = Clustering(AT, G);
   foreach (block G_i of G) {
5      D_Gi = GetEarliestΔ(AT, G_i);
       foreach (gate g in G_i)
6          M = ReplaceGate(M, L, g)
7      F_Gi = ComputeControlCirc(M, D_Gi, T);
8      M = AddControlCirc(M, G_i, F_Gi);
   }
}
```

Fig. 6. Clustered gate freezing algorithm.

glitching-capacitance product, gc for brevity) and to select the first $N$ gates of the sorted list.

Glitching is measured by counting the number of transitions at the output of each gate obtained by real delay simulation and by subtracting from this amount the number of transitions obtained through zero-delay simulation. The latter values must either be one or zero, depending on whether there is a transition or not. This difference represents the spurious transitions propagating through a gate.

Nodes with slack smaller than the delay increase caused by the replacement of a standard gate with a *F-gate* cannot be selected because replacing them with *F-Gates* would slow down the circuit; consequently, node selection is applied only to gates with nonzero slack (more precisely, slack larger than the delay increase caused by *F-gates*). In addition, we observed that picking new gates in the recursive fanout of already chosen gates may be disadvantageous. In fact, the presence of an *F-Gate* in the fanin cone of a gate tends to reduce the glitching activity of the gate itself. This effect could be taken into account by performing power analysis after selecting each node; however, this could make the optimization slow in case of large circuits.

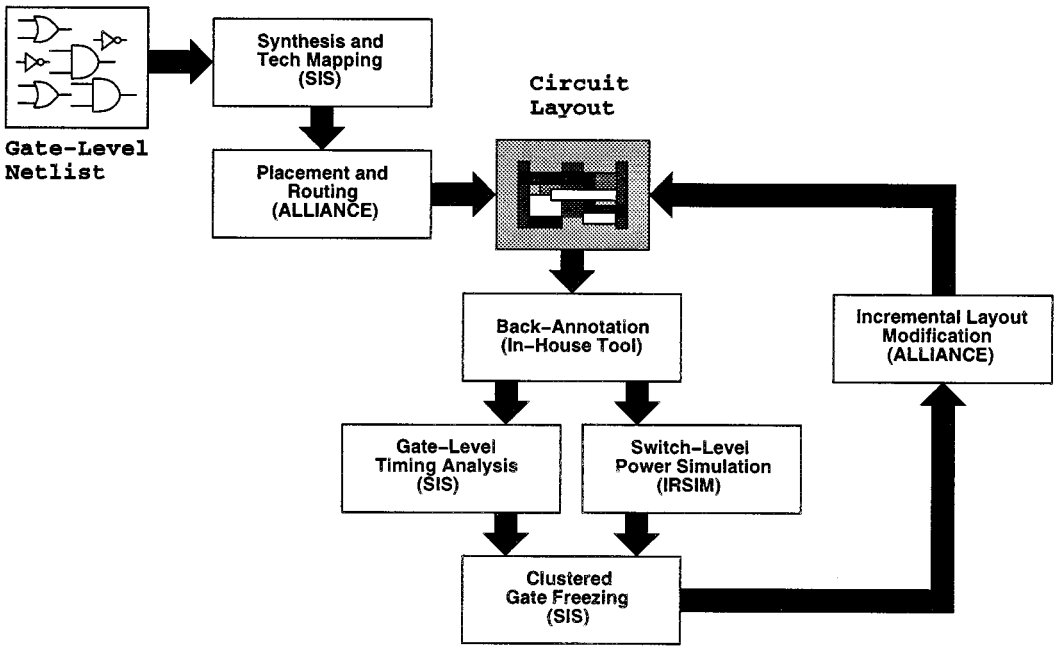



Fig. 7. Experimental environment.

TABLE I
EXPERIMENTAL RESULTS

| *Circ* | *PI* | *PO* | *Gates* | *Original* | | | *Optimized* | | | [%] | | | *Clust. Heur.* | *K* | *N* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | *GP* | *TP* | *Area* | *GP* | *TP* | *Area* | *GP* | *TP* | *Area* | | | |
| c432 | 37 | 6 | 312 | 0.34 | 1.44 | 829290 | 0.30 | 1.38 | 853339 | -10.2 | -4.6 | +2.9 | Growth | 4 | 30 |
| c499 | 41 | 32 | 637 | 1.33 | 4.09 | 5643849 | 1.02 | 3.86 | 5818808 | -23.5 | -5.6 | +3.1 | Growth | 5 | 30 |
| c880 | 60 | 26 | 479 | 0.81 | 2.92 | 2130128 | 0.71 | 2.76 | 2153559 | -14.1 | -5.7 | +1.1 | Growth | 4 | 30 |
| c1355 | 41 | 32 | 755 | 2.82 | 6.82 | 2846062 | 2.66 | 6.55 | 2908675 | -6.1 | -4.1 | +2.2 | Growth | 6 | 40 |
| c1908 | 33 | 25 | 854 | 3.51 | 9.13 | 4079104 | 3.17 | 8.64 | 4201477 | -10.6 | -5.6 | +3.0 | Match | 6 | 40 |
| c2670 | 233 | 139 | 1036 | 5.38 | 12.58 | 5319860 | 4.28 | 11.30 | 5355734 | -16.7 | -10.1 | +4.1 | Match | 8 | 50 |
| c3540 | 50 | 22 | 1587 | 13.11 | 30.01 | 13447900 | 9.90 | 25.34 | 13703410 | -24.4 | -15.5 | +1.9 | Match | 5 | 70 |
| c5315 | 178 | 123 | 2959 | 23.29 | 50.85 | 42937951 | 21.76 | 52.41 | 43925524 | -6.5 | +2.9 | +2.3 | Match | 10 | 80 |
| c6288 | 32 | 32 | 4354 | 104.27 | 256.62 | 69175176 | 90.05 | 232.07 | 71942183 | -13.7 | -9.6 | +3.8 | Match | 10 | 90 |
| c7552 | 207 | 108 | 3753 | 19.39 | 40.52 | 60352334 | 16.98 | 36.96 | 61921495 | -12.4 | -8.8 | +2.6 | Match | 9 | 80 |
| alu2 | 10 | 6 | 457 | 0.89 | 3.34 | 1842912 | 0.73 | 3.05 | 1887141 | -17.3 | -8.7 | +2.4 | Growth | 4 | 30 |
| alu4 | 14 | 8 | 1010 | 3.48 | 11.79 | 6284460 | 2.60 | 10.54 | 6403864 | -25.2 | -10.6 | +1.9 | Growth | 8 | 50 |
| dalu | 75 | 16 | 1492 | 5.02 | 17.35 | 12093812 | 4.09 | 17.06 | 12601752 | -18.4 | -2.3 | +3.4 | Match | 9 | 70 |
| frg2 | 143 | 139 | 1346 | 1.91 | 12.34 | 9731865 | 1.80 | 11.81 | 10062748 | -5.7 | -4.3 | +3.3 | Growth | 9 | 70 |
| t481 | 16 | 1 | 1089 | 0.90 | 7.78 | 7490340 | 0.86 | 7.61 | 7722540 | -4.7 | -2.2 | +4.2 | Match | 8 | 50 |
| **Avg.** | | | | | | | | | | -14.0 | -6.3 | +2.8 | | | |

We have adopted an enhanced selection procedure that approximately takes into account previous selections. Initially, all nodes are marked with an integer label, which is set equal to zero. Then, the node with highest glitching-capacitance product is selected. If two or more nodes have the same gc value (with a 10% tolerance margin), we use the total capacitive load of the transitive fanout as a tie breaker. The gate with highest total capacitive load of the transitive fanout is chosen first. The label of all the nodes in the transitive fanout of the selected node is then incremented. Node labels are used as tie breaker for successive selections. If two or more nodes have the same gc value (with the usual 10% tolerance margin), we select the one with smallest label. If this rule is not sufficient to break all the ties, the total capacitive load of the transitive fanout is used. If some ties persist, we choose randomly. The process is stopped after $N$ nodes have been selected.

### B. Clustering

In the algorithm of Fig. 5, one control signal for each selected gate needs to be generated. Although the number of gates replaced is usually a small fraction of the cells that are present in the circuit implementation (less than 5%), the generation of a control signal for each *F-Gate* is still quite impractical, since each signal has generally a duty cycle value different from any other, and there is then little chance of exploiting some sharing.

To reduce both the area of the control circuitry implementation and the routing of the control signals, we need to limit the number of such signals as much as possible. One way of doing

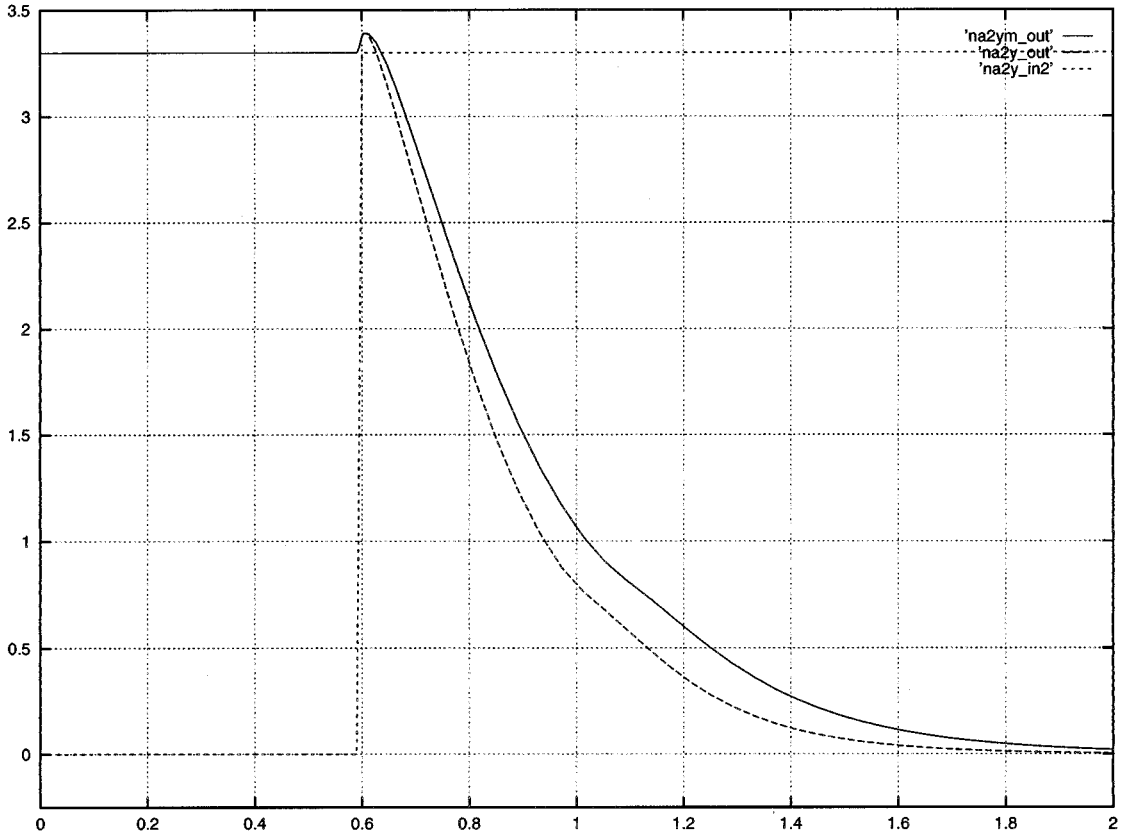



Fig. 8. Response to a rising input transition.

this is to *cluster* the *F-Gates* according to the values of their arrival times $\Delta$'s, as defined in (1).

The clustering problem can be stated as follows. Given a set of $N$ gates $G = (g_1, \cdots, g_N)$ and their arrival times $\Delta_{g_1}, \cdots, \Delta_{g_N}$, find a partition $\mathcal{P} = (P_1, \cdots, P_K)$ of the $\Delta$'s such that the partition is balanced, the variance of the $\Delta$ times within each block $P_i$ of the partition is minimized, and the number of blocks $K$ is small.

The partition $\mathcal{P}$ on the arrival times induces a partition on the set of selected gates $\mathcal{G} = (G_1, \cdots, G_K)$.

The requirement of the minimum variance is related to the error that this approximation introduces. In fact, all the gates belonging to the same block $G_i$ will be fed by the same control signal, whose delay, with respect to the clock, is determined by the earliest of their $\Delta$ times. More formally, the delay $D_{G_i}$ for all the control signals of the gates in $G_i$ is given by

$$D_{G_i} = \min_{g \in G_i} \Delta(g) \tag{2}$$

where $\Delta(g)$ is defined as in (1).

The approximation introduced by clustering some of the control signals together arises from the fact that all the gates $g$ in $G_i$ having $\Delta(g)$ larger than the $D_{G_i}$ will allow the propagation of the glitches occurring in the time interval $[\Delta(g_{\min}), \Delta(g)]$, where $g_{\min}$ is the gate in $G_i$ that determines the bound of (2).

Fig. 6 shows the pseudocode of the clustering-based gate freezing algorithm.

The flow of this algorithm is similar to that of Fig. 5. The main difference stands in Lines 4–8, where a proper partition of the candidate gates is first built (Line 4) (the details on the clustering procedure are described in Section IV-C). Procedure `Clustering` returns a partition $\mathcal{P} = (P_1, \cdots, P_K)$ of the $\Delta$ times and the corresponding partition of the selected gates $\mathcal{G} = (G_1, \cdots, G_K)$. Then, the algorithm iterates over the $K$ blocks of $\mathcal{G}$, and derives, for each cluster $G_i$, the values of $D_{G_i}$, as defined by (2) (Line 5). Each gate $g \in G_i$ is then replaced by the corresponding *F-Gate* (Line 6). After all the gates in $G_i$ have been replaced, the circuitry for generating the shared control signal is determined (Line 7) and incrementally added to the circuit layout (Line 8).

### C. Clustering Heuristics

The clustering problem, as stated in Section IV-B, is too general. In fact, the optimal solution may require a number of clusters, $K$, which may be too large. For particular distributions of the $\Delta$ values, the optimal solution can degenerate into the case of $K = N$ clusters of size 1. Therefore, we actually solve a constrained problem, where $K$ is upper bounded by a user-specified value.

We have implemented two heuristics that solve the clustering problem when $K$ is upper bounded by a user-specified value. Experiments have shown that the first one, based on clusters growth, works better on clusters of small cardinality (and thus on small circuits), while the second heuristics, based on matching, best performs on mid- to large-sized clusters (and thus on mid to large circuits).

In both cases, the cost function $\sigma_{\text{avg}}$ we have used to drive the clustering procedure is the average cluster variance (i.e.,

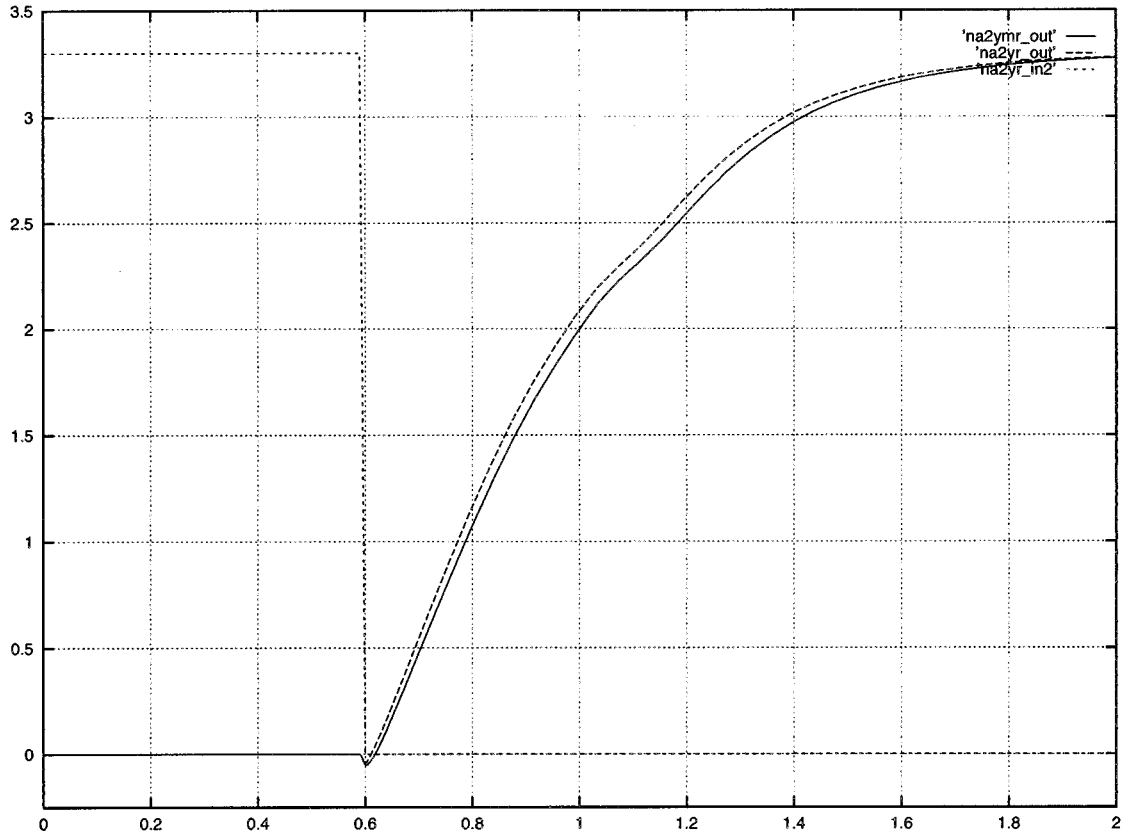



Fig. 9. Response to a falling input transition.

the average $\Delta$ of the gates in $G_i$), whose formal definition is the following [we assume to have a total of $K$ clusters, $(G_1, \cdots, G_i, \cdots, G_K)$]:

$$\sigma_{\text{avg}} = \frac{\sum_i \sigma_i}{K}$$

where

$$\sigma_i = \frac{\sum_{g \in G_i} (\Delta(g) - \Delta_{\text{avg}})^2}{|G_i|}$$

is the variance of the $\Delta$ values in cluster $G_i$.

Obviously, lower values of $\sigma_{\text{avg}}$ identify better solutions.

*1) Cluster Growth Heuristics:* The first clustering heuristics, that we call *cluster growth*, is based on a greedy algorithm that starts with a nonpartitioned set of nodes, and places them into a given number of clusters according to some affinity measure.

The algorithm consists of two phases: The first step is the selection of the *seed* nodes for each cluster, that is, the nodes that are placed first in each cluster. Seed selection affects the quality of the final solution, and several heuristic choices are possible. In our implementation, we select as seed the node having a $\Delta$ value closest to the average $\Delta$ of the nodes still to be assigned. Intuitively, this choice selects the node that attracts nodes to itself with a minimum error. The second step is the assignment of the nodes to clusters, which is carried out by selecting the nodes with minimum $\Delta$ "distance" from the nodes already belonging to a cluster. For a generic unassigned node n, we define its closeness $c_{n,i}$ to a cluster $G_i$ as follows ($j$ are the nodes in $G_i$):

$$c_{n,i} = \left| \sum_{j \in G_i} \Delta(j) - \Delta(n) \right|.$$

Notice that, in our implementation, the $K$ clusters are built sequentially one at a time. The procedure terminates when all the nodes have been assigned and may leave the last cluster with less nodes than the value of the bound.

This algorithm has a $O(N)$ complexity, but it generally produces suboptimal results, especially when the distribution of the values increases, since the affinity measure of a node is computed only with respect to nodes already assigned to the clusters. Better solutions, yet with an increased complexity, can be obtained by either evaluating the affinity measure of a node also with respect to not yet assigned nodes, or by collapsing clustered nodes into a supernode and considering the supernodes as single objects in the next clustering steps (*hierarchical* clustering).

*2) Matching Heuristics:* The second heuristics we introduce exploits existing graph algorithms to solve the node clustering problem.

Given the set of candidate nodes in the circuit $G = (g_1, \cdots, g_N)$, and the set of their corresponding arrival times $\Delta = (\Delta_1, \cdots, \Delta_N)$, we build a complete weighted graph $S = (V, E, W)$, where $V \equiv G$ is the set of vertices, $E = \{e_{ij}\}$ is the set of edges, and $W = \{w_{ij}\}$ is the set of edge weights, defined as $w_{ij} = |\Delta_i - \Delta_j|$.

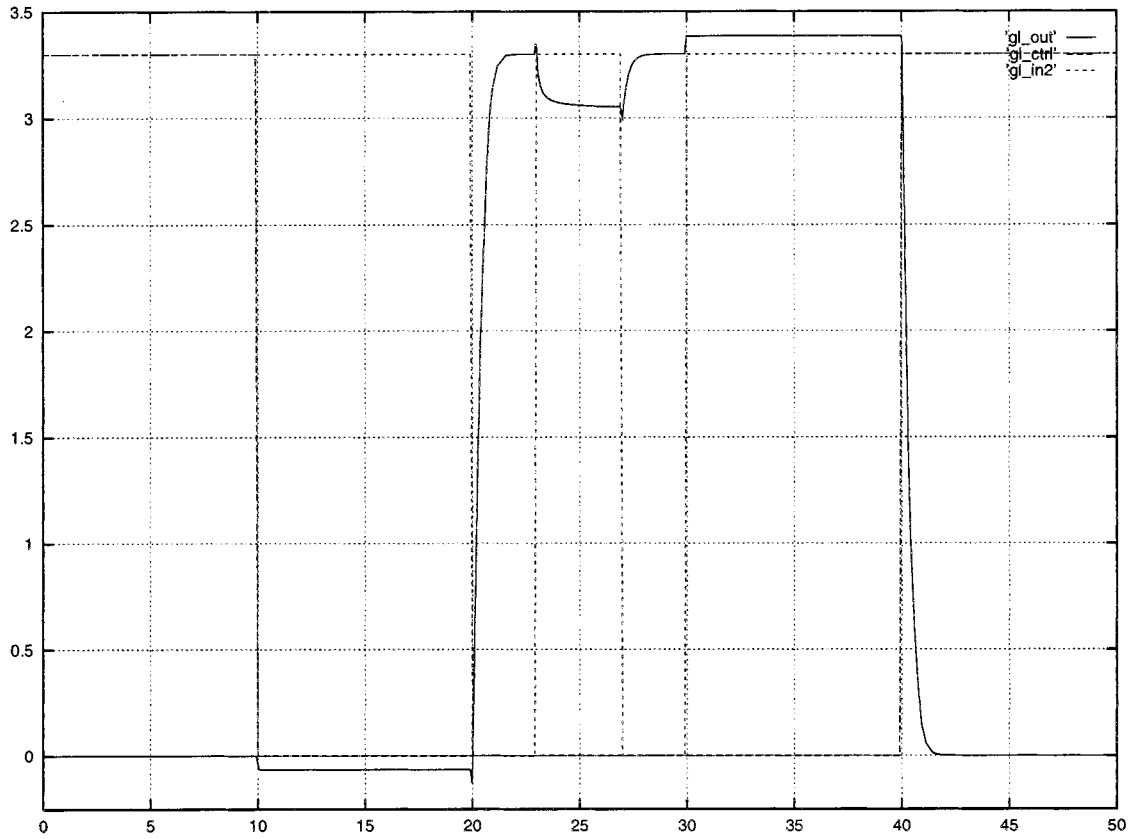



Fig. 10. Response to some input glitches.

The edge weights represent the skew in the values of $\Delta$ for each gate pair. Intuitively, we want to grow clusters with the smallest overall variance of $\Delta$. An interesting result from graph theory [16] shows that this process can be accomplished by iteratively clustering *pairs* of nodes.

The clustering problem for a bound size of two is equivalent to a *maximum weighted matching* [17]. In [16], it is shown that it is possible to solve, with bounded error, the clustering problem for a generic cluster size limit $B$, as follows.

- Find a maximum weighted matching on the graph.
- Pick $B/2$ edges at a time from the matching (in any order) to form a cluster of size $B$.

Adapting this algorithm to our clustering problem requires some minor modifications. First, given that the edge weights represent a differential quantity, we must replace the maximum weighted matching with a *minimum* weighted matching. The underlying algorithm we have used is the one of [18], which has a complexity of $O(N^3)$.

Second, in our case, the order of edge selection from the matching is not irrelevant. In fact, once we have identified pairs with smallest difference in $\Delta$, we must group matched pairs that have close *absolute* $\Delta$ values.

To achieve such selective clustering, we simply need to sort the edges in the matching in increasing (or decreasing) order of their *average* $\Delta$ and sequentially pick pairs according to the resulting order. The $\Delta_{\text{avg}}$ of a node pair is simply defined as the average of the $\Delta$'s of the corresponding nodes. This additional sorting step does not increase the complexity of the algorithm, since the $O(N^3)$ cost of the matching still dominates.

## V. Experimental Results

We have implemented the clustered gate freezing procedure using SIS [19] as gate-level front–end. The experimental environment we employed is depicted in Fig. 7.

The initial circuits were optimized using `script.delay`, and mapped using `map-n1-AFG` onto a gate library consisting of two- and three-input Nand and Nor gates and Inv and Buf gates with three different driving capabilities. Placement and routing of the tech-mapped circuits onto a 0.6-$\mu$m static CMOS physical-level library was done using Alliance [20]. Gate- and switch-level netlists were extracted from the layout using an in-house tool; such netlists were used for gate-level timing analysis (using SIS) and for switch-level power simulation (using Irsim [21]). After applying clustered gate freezing, the layouts were incrementally modified using Alliance, and the power dissipated by the optimized circuits was estimated (using Irsim) on the switch-level netlists extracted from the final layouts. Timing verification was also performed using SIS on the gate-level netlist derived from the final layout.

The circuits we considered for the experiments are the Iscas'85 benchmarks [4]; results for some of the largest Mcnc'91 multilevel networks are also reported [5].

Table I summarizes the data. In particular, columns *PI*, *PO*, and *Gates* report the structural characteristics of the examples. Columns *GP*, *TP*, and *Area* give the glitch power (in milliwatts),

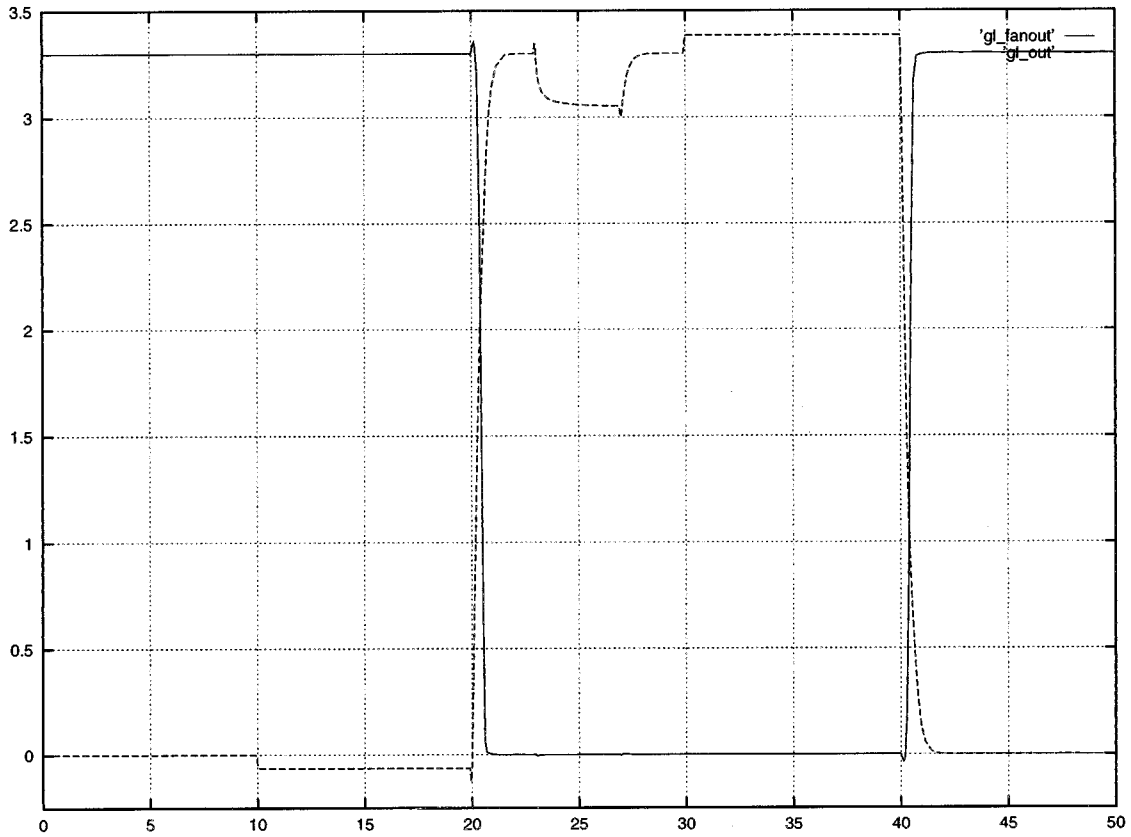



Fig. 11. Response of an inverter on the output of the *F-Nand*.

the total power (in milliwatts), and the layout area (in $\lambda^2$) before (columns *Original*) and after (columns *Optimized*) optimization. Columns [%] give the percentages of power and area variation. The three right-most columns indicate the clustering heuristics, the number of clusters $(K)$, and the number of frozen gates $(N)$ that have been chosen for each experiment.

Results are promising. In fact, an average glitch power reduction of 14.0%, yielding in an average reduction of the total power of 6.3% has been achieved at the cost of a negligible area increase (2.8% on average). Obviously, the speed of the circuits has not changed, since only noncritical gates have been replaced with *F-Gates*.

Clearly, gate freezing exclusively targets glitch power reduction. Therefore, the quality of the results would only be marginally affected by gate-level optimization techniques (e.g., POSE [22]) that minimize zero-delay power.

## VI. Implementation of F-Gates

The savings in glitch power we have presented in Section V are promising, and they have been measured through switch-level simulation of the transistor netlist extracted from the final circuit layout. However, in the experiments it was optimistically assumed that *F-Gates* can be implemented with marginal performance degradation with respect to standard CMOS gates. In a realistic design environment, this assumption should be validated by an accurate analysis of the impact that *F-Gates* may have on the circuit performance and reliability.

In this section, we carry out a detailed experimental investigation of the characteristics of *F-Gates*. More specifically, we have designed and generated the layout of some *F-Gates*, and we have studied how the behavior of such cells differs from that of the corresponding cells of a standard CMOS library. In the following, we present the results referred to a two-input Nand gate.

The *F-Gates* have been generated using the layout editor available in Alliance, and the simulation results have been collected through Spice-3f4 [23].

Our target during the realization of the two-input *F-Nand* cell was to guarantee, when the gate operates normally, an increase in the worst case delay below 20% of the value of the corresponding CMOS gate. This has been obtained by enlarging the size of the n transistors by approximately a factor of 1.5 with respect to the standard CMOS cell. The total gate area of the *F-Nand* amounts to $1260\lambda^2$, against an area of $756\lambda^2$ of the traditional CMOS implementation. The area increase is thus relevant. Notice, however, that such increase is mainly due to the accommodation of the control input at the boundaries of the *F-Gate*, rather than to the insertion of the additional pull-down transistor or to the upsizing of the existing n transistors. The increase in active area is in fact limited, and all confined into the n network (from $208\lambda^2$ to $440\lambda^2$).

Figs. 8 and 9 present the output response of a traditional CMOS gate and that of the *F-Nand* (in normal operation mode) to a rising and a falling transition, respectively, occurring on the slowest of the inputs. From the diagrams it can be easily observed how the *F-Gate* is more penalized, in terms of delay,

when the one–zero transition occurs on the output (from 239 to 287 ps) than in the opposite situation (from 285 to 300 ps). This was expected because the path that connects the output to ground (chain of n transistors) is longer than the one which goes from the output to the $V_{\rm dd}$ (p network).

We now consider the situation in which the *C-Signal* is active. In the ideal case, any one–zero transition would be prohibited on the output of the *F-Gate*. Unfortunately, in reality, when the gate is frozen and the pull-up transistors are off, the output node is floating (i.e., it is in the high-impedance state); therefore, it is prone to capacitive coupling that may generate some small "bumps" under or over the logic 0 or 1 values. This undesirable behavior is illustrated in Fig. 10, where the response of the two-input *F-Nand* to a series of glitches occurring on the slowest input is plotted. Obviously, the larger the load the *F-Gate* must drive, the smaller the bumps.

Fortunately, the bumps appearing at the output of the *F-Nand* are not strong enough to propagate through gates in the cell's fanout. In other words, the nominal voltage levels are restored as soon as the output signal of the *F-Gate* travels through the loading gates. This situation is experimentally shown in the diagram of Fig. 11, where the response of an inverter placed at the output of the *F-Nand* is plotted.

We have designed and simulated approximately 20 *F-Gates* implementing different functions, having an increasing number of inputs and different driving strengths. We have observed that Nor-like gates usually behave better than Nand-like cells, in terms of both output response in normal conditions and presence of over/under voltage levels in response to glitches when the *C-Signal* is active. This is due to the fact that the chain of n transistors is shorter for this kind of gate than for Nand-like cells.

In conclusion, we can state that *F-Gates* can be used in practice as glitch filtering devices. In fact, delay increase can be kept under control by properly resizing the transistors of the n net, and the area overhead, although noticeable, does not hamper the applicability of the method because only a few *F-Gates* (less than 5% of the total) are inserted in the layout for optimization purposes.

## VII. Conclusion

We have proposed a technique for glitch power minimization in CMOS circuits. The method consists of replacing some high-glitching gates with devices that are able to filter out spurious transitions whenever a proper control signal is activated. The technique has the distinctive feature of being applicable after layout, since it performs *in-place* optimization on the placed and routed description, and it only requires incremental rewiring.

## References


[1] L. Benini, M. Favalli, and B. Riccó, "Analysis of hazard contributions to power dissipation in CMOS ICs," in *IWLPD-94: ACM/IEEE Int. Workshop Low Power Design*, Apr. 1994, pp. 27–32.

[2] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd ed. Reading, MA: Addison-Wesley, 1992.

[3] L. Lavagno, P. McGeer, A. Saldanha, and A. Sangiovanni Vincentelli, "Timed Shannon circuits: A power-efficient design style and synthesis tool," in *DAC-32: ACM/IEEE Design Automation Conf.*, June 1995, pp. 254–260.

[4] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran," in *ISCAS-85: IEEE Int. Symp. Circuits and Systems*, June 1985, pp. 785–794.

[5] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Microelectronics Center of North Carolina Tech. Rep., Jan. 1991.

[6] L. Lavagno and A. Sangiovanni Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits*. Norwell, MA: Kluwer Academic, 1993.

[7] U. Ko, P. T. Balsara, and W. Lee, "A self-timed method to minimize spurious transitions in low power CMOS circuits," in *ISLPE-94: IEEE Int. Symp. Low Power Electronics*, Sept. 1994, pp. 62–63.

[8] E. Musoll and J. Cortadella, "Low power array multipliers with transition-retaining barriers," in *PATMOS-95: Int. Workshop Power and Timing Modeling, Optimization and Simulation*, Oct. 1995, pp. 227–238.

[9] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *ICCAD-93: ACM/IEEE Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 398–402.

[10] A. Raghunathan, S. Dey, and N. Jha, "Glitch analysis and reduction in register transfer level power optimization," in *DAC-33: ACM/IEEE Design Automation Conf.*, Jun. 1996, pp. 331–336.

[11] M. Hashimoto, H. Onodera, and K. Tamaru, "A power optimization method considering glitch reduction by gate sizing," in *ISLPED-98: ACM/IEEE Int. Symp. Low Power Electronics and Design*, Nov. 1998, pp. 221–226.

[12] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: Pushing power management to logic synthesis/design," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 1051–1060, Nov. 1998.

[13] L. Benini and G. De Micheli, "Transformation and synthesis of FSM's for low power gated clock implementation," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 630–643, June 1996.

[14] G. Kim, M.-K. Kim, B.-S. Chang, and W. Kim, "A low-voltage, low-power CMOS delay element," *IEEE J. Solid-State Circuits*, vol. 31, pp. 966–971, July 1996.

[15] L. Benini, P. Vuillod, A. Bogliolo, and G. De Micheli, "Clock skew optimization for peak current reduction," *Kluwer J. VLSI Signal Processing*, vol. 16, no. 2/3, June 1997.

[16] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. New York: Wiley, 1990.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *An Introduction to Algorithms*. New York: McGraw-Hill, 1990.

[18] H. Gabow, "An efficient implementation of Edmonds algorithm for maximum matching on graphs," *J. ACM*, vol. 23, no. 2, pp. 221–234, Apr. 1976.

[19] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuits design using synthesis and optimization," in *ICCD-92: IEEE Int. Conf. Computer Design*, Oct. 1992, pp. 328–333.

[20] A. Grenier and F. Pecheux, "ALLIANCE: A complete set of CAD tools for teaching VLSI design," Univ. Pierre e Marie Curie Tech. Rep., Paris, France, 1993.

[21] A. Salz and M. Horowitz, "IRSIM: An incremental MOS switch-level simulator," in *DAC-26: ACM/IEEE Design Automation Conf.*, June 1989, pp. 173–178.

[22] S. Iman and M. Pedram, "POSE: Power optimization and synthesis environment," in *DAC-33: ACM/IEEE Design Automation Conf.*, June 1996, pp. 21–26.

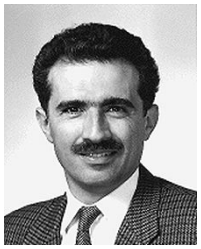
[23] L. W. Nagel, "SPICE-2: A computer program to simulate semiconductor circuits," Univ. Calif. Memo ERL-M520, Berkeley, 1975.



**Luca Benini** (M'93) received the Dr.Eng. degree in electrical engineering from the University of Bologna, Bologna, Italy, in 1991 and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1997, respectively.

Since 1998, he has been an Assistant Professor in the Department of Electronics and Computer Science, University of Bologna. He also holds visiting professor positions at Stanford University and Hewlett-Packard Laboratories, Palo Alto, CA. His research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications and in the design of portable systems.

Dr. Benini has been a member of technical program committees for several technical conferences, including the Design and Test in Europe Conference and the International Symposium on Low Power Design.



**Giovanni De Micheli** (F'94) is a Professor of Electrical Engineering and Computer Science at Stanford University, Stanford, CA. His research interests include several aspects of the computer-aided design of integrated circuits and systems, with particular emphasis on automated synthesis, optimization, and validation. He is the author of *Synthesis and Optimization of Digital Circuits* (New York: McGraw-Hill, 1994) and a coauthor of *Dynamic Power Management: Circuit Techniques and CAD Tools* (Norwell, MA: Kluwer, 1998) and three other books. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN.

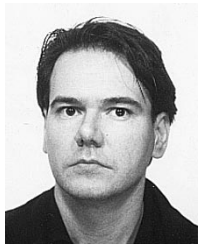
Dr. De Micheli received the 1987 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN/ICAS Best Paper Award, a Presidential Young Investigator Award in 1988, and two Best Paper Awards at the Design Automation Conference in 1983 and in 1993. He is Vice President (for publications) of the IEEE CAS Society. He is the General Chair of the 37th Design Automation Conference. He was Program and General Chair of the International Conference on Computer Design (ICCD) in 1988 and 1989, respectively. He was also Codirector of the NATO Advanced Study Institutes on Hardware/Software Co-design, Tremezzo, Italy, in 1995 and the Logic Synthesis and Silicon Compilation, L'Aquila, Italy, in 1986.

**Alberto Macii** (S'00) received the Dr.Eng. degree in computer engineering from the Politecnico di Torino, Torino, Italy, in 1996. He is currently working toward the Ph.D. degree in computer engineering at the Politecnico di Torino.

His research interests include several aspects of the development of algorithms, methods, and tools for low-power digital design.

**Enrico Macii** (M'92) received the Dr.Eng. degree in electrical engineering from the Politecnico di Torino, Torino, Italy, in 1990, the Dr.Sc. degree in computer science from the Università di Torino, Torino, in 1991, and the Ph.D. degree in computer engineering from the Politecnico di Torino in 1995.

From 1991 to 1994, he was an Adjunct Faculty Member at the University of Colorado, Boulder. Currently, he is an Associate Professor at the Politecnico di Torino. His research interests include synthesis, verification, simulation, and testing of digital circuits and systems. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN.



**Massimo Poncino** (M'97) received the Dr.Eng. degree in electrical engineering in 1989 and the Ph.D. degree in computer engineering in 1993, both from the Politecnico di Torino, Torino, Italy.

From 1993 to 1994, he was a Visiting Faculty Member at the University of Colorado, Boulder. Currently, he is an Assistant Professor at the Politecnico di Torino. His research interests include synthesis, verification, simulation, and testing of digital circuits and systems.

**Riccardo Scarsi** received the Dr.Eng. degree in electrical engineering from the Politecnico di Torino, Torino, Italy, in 1997. He is currently working toward the Ph.D. degree in computer engineering at the Politecnico di Torino.

His research interests include several aspects of the development of algorithms, methods, and tools for low-power digital design.