Glitch Power Minimization by Gate Freezing

L. Benini[◇] G. De Micheli [#] A. Macii[‡] E. Macii[‡] M. Poncino[‡] R. Scarsi[‡]

[‡] Politecnico di Torino
Torino, ITALY 10129

[#] Stanford University
Stanford, CA 94305

[◇] Università di Bologna
Bologna, ITALY 40136

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. 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.

1 Introduction

Minimizing glitching at the gate level is a complex task, because it is difficult to estimate the impact that the 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 (e.g., Shannon [1] circuits); alternatively, a non-glitch-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 an unpredictable fashion.

We follow the second avenue of attack, 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 re-wiring of a few signal nets. The cost function that drives the optimization is very accurate, because glitch and total power estimation are carried out on a network with accurate back-annotation of wiring loads.

Automatic techniques for glitch minimization in logic circuits have been the subject of intensive research in the past. Some of them have been developed for specific applications [4], while others are general-purpose [5, 6]. In addition, *guarded evaluation* [7], although not explicitly targeting glitch power minimization, has some affinity with the gate freezing idea we present in this paper. Differently from gate freezing, the techniques above are applied before placement and routing.

We have applied the gate freezing procedure to the Iscas'85 benchmarks [2], as well as to some examples taken from the Mcnc'91 suite [3]. We have achieved an average glitch power reduction of 14.0%, resulting in an average total power savings of 6.3%. Area and speed of the circuits are virtually 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 balance.

2 Gate Freezing

The flow of the gate freezing procedure starts from a placed and routed combinational circuit. We assume that wire loads have been extracted and back-annotated into the gate-level model of the circuit. Accurate switch-level simulation of user-specified patterns is performed to obtain a detailed statistic 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 on timing constraints is subsequently exploited in the gate-selection step. A fraction of the gates in the netlist is selected. We choose non-critical gates with high glitching and high fanout. 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 subsections 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 3.

2.1 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 Figure 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.

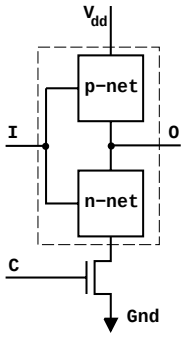

Figure 1: Basic Transformation of a Library Cell.

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 1 to 0 transition is actually forbidden, whereas the gate output can still exhibit the 0 to 1 transition. This may occur for any input configuration that is supposed to force a 1 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.
The above considerations imply 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, $\overline{C}$, of the control input C.
This solution would clearly suppress any transition on the gate output, and solve the above limitation of the single n-transistor solution. However, there are some considerations that make the introduction of the p-transistor undesirable: *i*) The load on the signal C is doubled for any gate such signal must drive; *ii*) Both phases of signal C are required; *iii*) 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.

2.2 C-Signal Generation

As described at the beginning of this section, static timing analysis is carried out on the initial mapped circuit in a pre-processing phase. For each gate g, arrival time $AT(g)$, required time $RT(g)$, 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 3.1, 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 non-null 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 $AT(g)$ its arrival time. Moreover, let $X = (x_1, \ldots, x_n)$ denote the inputs of g, and $AT(x_i)$, $RT(x_i)$, $i = 1, \ldots, n$ 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 0 in the time interval between the beginning of the clock cycle and the time Δ that equals the latest arrival time of the inputs of g. In symbols:

$$\Delta(g) = \max_{i=1,\ldots,n} AT(x_i) \qquad (1)$$

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 non-critical, all of its inputs will be non-critical as well. Therefore, all the inputs of g will have some slack. What we must then guarantee is that C goes to 1 (i.e., g is free to evaluate) *before* the time , which is the earliest *required* time of all inputs of g. In symbols:

$$, (g) = \min_{i=1,\ldots,n} RT(x_i)$$

This allows a safety margin for the transition of the control signal, because we have a *don't care* region between $\Delta(g)$ and , (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 Figure 2. C can be thought of as a delayed copy of the clock, yet with a different duty cycle $D_C = \frac{\Delta(g)}{T}$. In the picture, the shaded area shows the slack for the 0 to 1 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 0 to 1 transition mentioned in Section 2.1. The shaded area in the figure shows this glitch-free portion of a clock cycle.

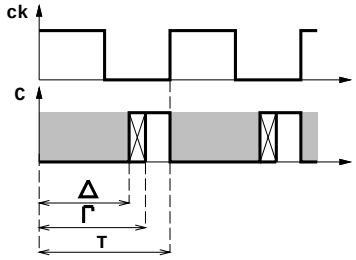

Figure 2: Relation Between Signal C and the Clock.

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 Δ's, and use the slack of the control signal , $-\Delta$ only as a safety margin.

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

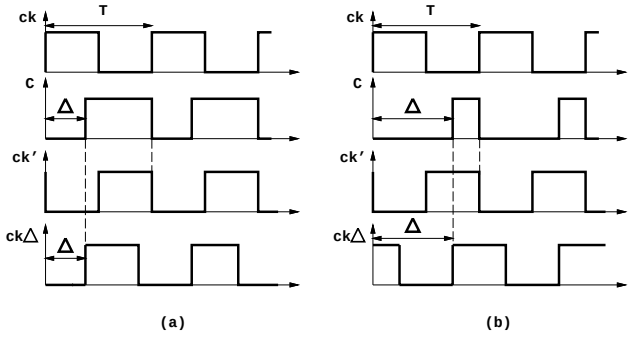

Figure 3: Timing Diagrams of Control Signal Generation.

Generating C requires a delayed version, ck_Δ, of the clock signal. The trivial implementation of a delay element is a chain of an even number of suitably sized inverters. Unfortunately, this implementation is highly power and area-inefficient when the delay is much larger than a single inverter delay. Several implementations have been proposed that overcome this limitation. We adopted the delay element proposed in [8], whose power dissipation is approximately three times that of a single inverter and area approximately 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.

3 Gate Selection and Clustering

The basic version of the gate freezing algorithm is shown in Figure 4. The procedure receives as inputs a mapped, placed and routed circuit, M, a library of cells, L, containing both ordinary cells and the corresponding *F-Gates*, and the maximum number *NFG* of gates to be selected for replacement with *F-Gates*. The procedure first performs static timing analysis (Line 1) and power estimation (Line 2); then, it selects candidate cells for replacement with *F-Gates* (Line 3). Finally, it replaces each selected gate g with the corresponding *F-Gate* (Line 4), it computes the timing for the control signal of such gate (Line 5), and it instantiates the circuitry for generating them (Line 6). We first analyze procedure `SelectCandidates` (Section 3.1), that returns the set of gates to be replaced by *F-Gates*. Then, we focus our attention on gate clustering (Sections 3.2 and 3.3), a key step that is required to make gate freezing applicable in practice.

```
procedure GateFreeze(M, L, NFG) {
1   (RT[ ], AT[ ]) = StaticTimingAnalysis(M);
2   Power[ ] = PowerSimulation(M);
3   G = SelectCandidates (M, Power, AT, NFG);
    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);
    }
}
```

Figure 4: Gate Freezing Algorithm.

3.1 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 take into account the capacitive load of the nodes. Therefore, our simple procedure for cell selection is to sort the network nodes in decreasing order of glitching activity weighted by load capacitance (we call this metric glitching-capacitance product, *gc* for brevity), and to select the first *NFG* gates of the sorted list. Glitching is measured by counting the number of transitions at the output of each gate obtained by accurate (i.e., real delay) simulation, and by subtracting from this amount the number of transitions obtained through ideal (i.e., zero-delay) simulation. The latter values must either be 1 or 0, depending on whether there is a transition or not. This difference represents the spurious transitions propagating through a gate.

Remember that nodes with zero slack cannot be selected, because replacing them with *F-Gates* would slow down the circuit; consequently, node selection is applied only to gates with non-zero slack. 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 process could make the optimization procedure very slow in the case of large circuits.

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 0. 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 the ties persist, we chose randomly. The process is stopped after NFG nodes have been selected.

3.2 Clustering

In the algorithm of Figure 4, 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 to exploit 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 this is to *cluster* the *F-Gates* according to the values of their arrival times Δ's, as defined in Equation 1. The clustering problem can be stated as follows: Given a set of NFG gates $G = (g_1, \ldots, g_{\mathrm{NFG}})$, and their arrival times $\Delta_{g_1}, \ldots, \Delta_{g_{\mathrm{NFG}}}$, find a partition $\mathcal{P} = (P_1, \ldots, P_K)$ of the Δ's such that: *i*) The partition is balanced; *ii*) The variance of the Δ times within each block P_i of the partition is minimized; *iii*) The number of blocks, K, is small.

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

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 Δ times. More formally, the delay D_{G_i} for all the control signals of the gates in G_i is given by:

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

where $\Delta(g)$ is defined as in Equation 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 $\Delta(g_i)$ 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 Equation 2.

Figure 5 shows the pseudo-code of the clustering-based gate freezing algorithm.

```
procedure ClusteredGateFreeze(M, L, NFG) {
1   (RT[ ], AT[ ]) = StaticTimingAnalysis(M);
2   Power[ ] = PowerSimulation(M);
3   G = SelectCandidates(M, Power, AT, NFG);
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);
    }
}
```

Figure 5: Clustered Gate Freezing Algorithm.

The I/O interface and the flow of this algorithm are similar to those of Figure 4. 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 3.3). Procedure `Clustering` returns a partition $\mathcal{P} = (P_1, \ldots, P_K)$ of the Δ times, and the corresponding partition of the selected gates $\mathcal{G} = (G_1, \ldots, G_K)$. 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 Equation 2 (Line 5). Each gate $g \in G_i$ is then replaced with 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).

3.3 Clustering Heuristics

The clustering problem, as stated in Section 3.2, is too general. In fact, the solution with optimal cost may require a number of clusters, K, which may be too large. 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 σ_{avg} we have used to drive the clustering procedure is the average cluster variance (i.e., the average Δ of the gates in G_i), whose formal definition is the following (we assume to have a total of K clusters, $(G_1, \ldots, G_i, \ldots, G_K)$):

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

where:

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

is the variance of the Δ values in cluster G_i. Obviously, better solutions are identified by lower values of σ_{avg}.

The details of the two heuristics mentioned above are omitted for space reasons; for further information, the interested reader can refer to [9].

Circ	PI	PO	Gates	Original			Optimized			[%]			Clust.	NC	NFG
				GP	TP	Area	GP	TP	Area	GP	TP	Area	Heur.		
c432	37	6	312	0.343	1.448	829290	0.308	1.381	853339	-10.2	-4.6	+2.9	Growth	4	30
c499	41	32	637	1.337	4.092	5643849	1.022	3.861	5818808	-23.5	-5.6	+3.1	Growth	5	30
c880	60	26	479	0.816	2.923	2130128	0.715	2.766	2153559	-14.1	-5.7	+1.1	Growth	4	30
c1355	41	32	755	2.827	6.827	2846062	2.666	6.553	2908675	-6.1	-4.1	+2.2	Growth	6	40
c1908	33	25	854	3.518	9.133	4079104	3.179	8.646	4201477	-10.6	-5.6	+3.0	Match	6	40
c2670	233	139	1036	5.380	12.582	5319860	4.282	11.304	5355734	-16.7	-10.1	+4.1	Match	8	50
c3540	50	22	1587	13.111	30.019	13447900	9.904	25.340	13703410	-24.4	-15.5	+1.9	Match	5	70
c5315	178	123	2959	23.290	50.853	42937951	21.766	52.417	43925524	-6.5	+2.9	+2.3	Match	10	80
c6288	32	32	4354	104.277	256.622	69175176	90.051	232.075	71942183	-13.7	-9.6	+3.8	Match	10	90
c7552	207	108	3753	19.392	40.527	60352334	16.987	36.960	61921495	-12.4	-8.8	+2.6	Match	9	80
alu2	10	6	457	0.891	3.342	1842912	0.736	3.051	1887141	-17.3	-8.7	+2.4	Growth	4	30
alu4	14	8	1010	3.481	11.794	6284460	2.609	10.541	6403864	-25.2	-10.6	+1.9	Growth	8	50
dalu	75	16	1492	5.022	17.358	12093812	4.097	17.064	12601752	-18.4	-2.3	+3.4	Match	9	70
frg2	143	139	1346	1.912	12.348	9731865	1.803	11.810	10062748	-5.7	-4.3	+3.3	Growth	9	70
t481	16	1	1089	0.903	7.783	7490340	0.860	7.612	7722540	-4.7	-2.2	+4.2	Match	8	50
Avg.										-14.0	-6.3	+2.8			

Table 1: Experimental Results.

4 Experimental Results

We have implemented the clustered gate freezing procedure using SIS as the gate-level front-end. A pictorial description of our experimental environment is shown in Figure 6.

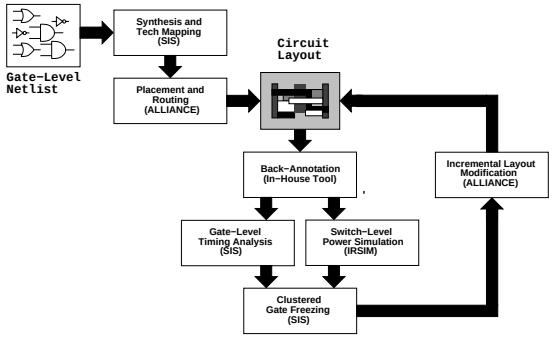

Figure 6: Experimental Environment.

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 [10]. 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 [11]). After applying the clustered gate freezing procedure, 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 [2]; results for some of the largest Mcnc'91 multi-level networks are also reported [3].

Table 1 summarizes the data. Columns *GP*, *TP*, and *Area* give the glitch power (in mW), the total power (in mW), and the layout area (in λ^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 (*NC*), and the number of frozen gates (*NFG*) that have been chosen for each experiment.

Results, though preliminary, are very promising. In fact, an average glitch power reduction of 14.0%, resulting 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 non-critical gates have been replaced with the (marginally slower) *F-Gates*.

Note that gate freezing exclusively targets glitch power reduction. Therefore, the quality of the results would only be marginally affected by gate-level power optimization techniques (e.g., POSE [12]) that minimize zero-delay power.

5 Conclusions

We have proposed a technique for glitch power minimization in combinational circuits. The method consists of transforming some high-glitching gates into modified devices that are able to filter out spurious output transitions whenever a proper control signal is activated. The technique has the distinctive feature of being applicable as a post-layout processing step, since it performs *in-place* optimization on the placed and routed description, and it only requires incremental re-wiring.

References

[1] L. Lavagno, P. McGeer, A. Saldanha, A. Sangiovanni Vincentelli, "Timed Shannon Circuits: A Power-Efficient Design Style and Synthesis Tool," DAC-32, Jun. 1995.

[2] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," ISCAS-85, Jun. 1985.

[3] S. Yang, Logic Synthesis and Optimization Benchmarks User Guide Version 3.0, Tech. Rep., MCNC, Jan. 1991.

[4] E. Musoll, J. Cortadella, "Low Power Array Multipliers with Transition-Retaining Barriers," PATMOS-95, Oct. 1995.

[5] J. Monteiro, S. Devadas, A. Ghosh, "Retiming Sequential Circuits for Low Power," ICCAD-93, Nov. 1993.

[6] A. Raghunathan, S. Dey, N. Jha, "Glitch Analysis and Reduction in Register Transfer Level Power Optimization," DAC-33, Jun. 1996.

[7] V. Tiwari, S. Malik, P. Ashar, "Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design," ISLPD-95, Aug. 1995.

[8] L. Benini, P. Vuillod, A. Bogliolo, G. De Micheli, "Clock Skew Optimization for Peak Current Reduction," Kluwer Journal of VLSI Signal Processing, Vol. 16, No. 2/3, Jun. 1997.

[9] L. Benini, G. De Micheli, A. Macii, E. Macii, M. Poncino, R. Scarsi, Glitch Power Minimization by Gate Freezing, Tech. Rep., Politecnico di Torino, Apr. 1998.

[10] A. Grenier, F. Pecheux, ALLIANCE: A Complete Set of CAD Tools for Teaching VLSI Design, Tech. Rep., Université Pierre e Marie Curie, 1993.

[11] A. Salz, M. Horowitz, "IRSIM: An Incremental MOS Switch-Level Simulator," DAC-26, Jun. 1989.

[12] S. Iman, M. Pedram, "POSE: Power Optimization and Synthesis Environment," DAC-33, Jun. 1996.