Post Sign-off Leakage Power Optimization

Hamed Abrishami[1], Jinan Lou[2], Jeff Qin[3], Juergen Froessl[3], Massoud Pedram[1]
Department of Electrical Engineering, University of Southern California, Los Angeles, CA[1]
Google Inc., Mountain View, CA[2]
Synopsys Inc., Mountain View, CA[3]
{habrisha, pedram}@usc.edu, jinan@google.com, {juergen, zqin@synopsys.com}

ABSTRACT

With the scaling down of the CMOS technologies, leakage power is becoming an increasingly important issue in IC design. There is a trade-off between subthreshold leakage power consumption and clock frequency in the circuit; i.e., for higher performance, leakage power consumption must be sacrificed and vice versa. Meanwhile, timing analysis during synthesis and physical design is pessimistic, which means there are some slacks available to be traded for leakage power minimization. This power minimization can be done after the sign-off which is more accurate and realistic than if it is done before the sign-off. The available slack can be traded for leakage power minimization by footprint-based cell swapping and threshold voltage assignment. In this paper, we introduce our post sign-off leakage power optimization problem as a nonlinear mathematical program and solve it by using conjugate gradient (CG) method. We set up a novel transformation technique to manipulate the constraints of the optimization problem to be solved by CG. We show that by doing this optimization we can reduce the leakage power consumption by 34% on average in comparison with no power optimization after sign-off. All experiments are done on the real industrial designs.

Categories and Subject Descriptors:

B.6.3: Design Aids-Optimization

General Terms: Algorithms, Design, Performance.

Keywords: Path-based analysis, Sign-off, Leakage power, Optimization, Slack, Conjugate gradient.

1. INTRODUCTION

As CMOS transistors are scaled toward ultra deep submicron technologies, the supply voltage is reduced so as to avoid device failure due to high electric fields in the gate oxide and the conducting channel under the gate. Voltage scaling reduces the circuit power consumption because of the quadratic relationship between dynamic power consumption and supply voltage, but it also increases the delay of logic gates. To compensate the resulting performance loss, transistor threshold voltages are decreased, which causes an exponential increase in the subthreshold leakage current. At deep sub-micron technology nodes, almost half of the total chip power consumption is due to leakage power [1].

The need for high performance computation in today's market forces the use of low threshold voltage transistors in the chips. Using this kind of transistors makes ICs power hungry even in the standby modes due to subthreshold leakage power consumption.

There is a trade-off between circuit performance (clock frequency) and leakage power; i.e., increasing performance causes the increase in leakage power and vice versa. To overcome this problem numerous sizing and threshold voltage assignments techniques in different phases of the design flow have been investigated [2][3][4][5]. All of these techniques basically trade the slack of non-critical paths for leakage minimization by doing high threshold voltage assignment and down-sizing the gates.

Although these techniques are useful in a typical design flow, no one so far has looked into the pessimism which is in the nature of the timing tools during the synthesis and physical design phases. During these phases timing of the circuits is analyzed using gated-based analyzers (GBA) which basically have linear complexity in terms of the number of the gates in the design. The GBA is pessimistic; i.e., the timing calculation is conservative. However, during sign-off more accurate timing tools, which make use of path-based analysis (PBA), are used. Hence, after sign-off there are often some positive slacks available in many paths of the circuit. These positive slacks can be traded for leakage power reduction.

The focus of this paper is to take advantage of these positive slacks and do post sign-off leakage power optimization considering delay constraints on the paths. This means no new timing violations are added to the circuit while the circuit power is reduced. The optimization is done by simultaneously changing version of each gate in the circuit. Different versions of each gate have the same footprint, in the sense that they have the same physical size and same locations for input and output pins. Consequently, there is no need to redo the placement and routing after changing the gate versions. Therefore, the problem is footprint-based swapping.

There are two types of footprint-based gate swapping, one is changing the transistor voltage thresholds within a gate, and the other is gate length biasing [6]. So, each gate version exhibits different leakage power consumption and switching speed.

To the best knowledge of the authors, this study is the first paper that addresses the post sign-off leakage power optimization taking advantage of precise timing information from PBA timer, and employs path-based circuit optimization. Previous works have all utilized GBA.

In [4] for example, the circuit is sized for maximum slack using all gates with low threshold voltage; timing slacks are allocated to gates based on their power/delay sensitivity factors. Next, each gate version to minimize leakage power subject to the given slack is found. Finally, the timing of the whole circuit is checked, if the circuit failed its delay target, the procedure would change the slack allocations and repeat the process. This loop continues until all timing requirements are met.

The problem with the above approach for our optimization is that after the sign-off we already have the sized circuit with the assigned threshold voltages. Moreover, effective slack allocation is a hard problem for large circuits [7] (hence the need to iterate many times, each time changing the slack allocation.) Finally, running the timer again to check for the violations considering the number of instances in today's ICs is really time expensive and can increase the turn-around time for many days.

The authors in [5] divide the circuit to multiple shells including the gates with equal slacks, and then trade the slack of each shell for lower leakage power. This technique suffers from the same problem as [4] since it again needs gate slack allocation.

Linear programming formulation has been used to exploit multiple-L_{gate} footprint-compatible cell libraries and post-layout L_{gate}-biasing to minimize the total leakage power under timing constraints (see [6].) The L_{gate}-biasing is used for the cells on non-critical paths. Although this paper is one of the few works that considers the post sign-off leakage power optimization, it still applies GBA and gate slack allocation in its optimization.

The remainder of this paper is organized as follows. Section 2 explains different methods of timing analysis. Section 3 elaborates the optimization problem and mathematical formulation to solve the problem. Section 4 gives details on the solution method to solve the optimization problem. The optimization flow is explained in section 5. Experimental results on real industrial test cases are shown in section 6 and section 7 concludes the paper.

2. BACKGROUND

The goal of timing analysis is to compute the signal arrival times at output nodes. Using these computed values and by knowing the required time at each output node, the slack for each path is determined. When the slack is negative, a timing violation has happened and the path is called the critical path.

2.1 Gate-Based Timing Analysis

Let directed graph *G(V, A)* represents the netlist of a circuit. The vertex set *V* is in one-to-one correspondence with the set of gates whereas the edge set *A* represents the pin-to-pin connections between gates. Associated with each gate g_i in the circuit, there exist a required arrival time r_i and an actual arrival time a_i. The arrival times for primary inputs and the required arrival times for primary outputs are specified by the designer of the circuit or obtained as a result of timing analysis at higher levels of the design.

The actual arrival time, a_j, is given by

$$a_j = \max\{(a_i + d_{i,j})\ \forall(v_i, v_j) \in A\}$$

The required arrival time, r_i, is given by

$$r_i = \min\{(r_j - d_{i,j})\ \forall(v_i, v_j) \in A\}$$

where $d_{i,j}$ is defined as the delay from node *i* to *j* which is basically the delay of g_i plus the delay of interconnect between g_i and g_j.

The slew (time) in each node is considered as the maximum of all input slews.

GBA is a useful method since it is linear in terms of the number of the gates in the circuit. It is used in the different stages of the VLSI design flow from synthesis to physical design. However, GBA introduces pessimism in the design, which tends to result in the over-design of the circuit.

There are two factors that contribute to pessimism in GBA:

1. In each node the maximum arrival time and maximum slew are propagated. Pessimism is created because the worst arrival time can be due to a circuit path (or timing event on a path) which does not result in the worst slew. Note also that the computed worst slew will be used to calculate the next gate's delay, which in turn causes a pessimistic arrival time calculation for the next node in the circuit.
2. GBA primarily relies on switching windows (between the minimum and maximum arrival times) of aggressors and victims for crosstalk calculation; even though, actual switching times are discrete. Hence, the timing windows are the major source of pessimism in crosstalk calculation [8].

2.2 Path-Based Timing Analysis

Definition 1: Each timing path is defined as a series of vertices and edges, which starts from a primary input or sequential cell output and ends at a primary output or sequential cell input.

Definition 2: Each primary output node or sequential cell input is called an **endpoint**.

Definition 3: The slack for each timing path is defined as

$$s_i = r_i - a_i\ \forall v_i \in endpoints$$

Path-based timing analysis (PBA) is more accurate than GBA since it is doing the timing calculation on paths rather than individual gates, which may lie on many different paths. Therefore, it does not have the pessimism factors mentioned in section 2.1. The major problem with the path-based analysis is that the number of paths in the circuit can be exponential.

In today's STA tools, first GBA is done. GBA determines the violated paths, and then PBA is done on these paths to obtain more accurate timing characteristics and remove the pessimism. By doing so, the number of timing violations can be reduced and the remaining ones are sent to the designers to be handled by changes in the design.

3. PROBLEM STATEMENT AND FORMULATION

The problem is to minimize the leakage power consumption in the circuit after sign-off. The paths that fail the timing requirements (have negative slacks) and the ones that have zero or positive slacks are known after the sign-off. The goal is to trade the positive slacks for lower leakage power, i.e., replace some gates in the paths with the positive slacks by versions of the gates that have lower leakage power consumption (but also lower switching speed.) These versions may consist of the high threshold voltage transistors or simply have slightly longer transistor channel lengths. The optimization should be done in a way that (i) paths with negative slacks do not become worse than before and (ii) no new paths with negative slacks are created.

We have to keep in our mind that gate sizing has already been done at this stage of the design flow and the circuit already includes gates with different threshold voltage assignments; i.e., low and high leakage power consuming gates. Each gate has *k* different versions that have different transistor channel lengths (and/or threshold voltage assignments with the same area footprint); hence, different leakage power consumptions and delays. Note that a gate also has other versions that have different channel widths (and hence drive strengths), but then they do not have the same footprint. For example, inverter-1x and inverter-2x are different versions of an inverter gate with different channel widths and for sure different footprints. These versions of a gate are different from the gate versions we talk about for the purpose of post-signoff leakage power saving. These different gate sizes have already been selected during design flow and will not be changed here. So, inverter-1x cannot be swapped by inverter-2x or vice versa and each one of them has their own individual versions (with same footprint) during post sign-off optimization.

We can also do some gate instance swapping in the negative slack paths to reduce the leakage power as long as we do not make the slack worse.

Timing constraints in this paper are path-based. The reason for this approach is that the real slack (not the pessimistic one) is available for each circuit path from a PBA timer (the sign-off timing analysis tool.) By applying the following problem formulation, we can simultaneously take care of gates that lie in multiple paths without optimizing each path separately. On the other hand, the gate sensitivity to the power and delay is taken into account without any need to calculate this factor separately. This gives us a chance to trade the power between power sensitive and power insensitive gates even in the paths with negative slacks. Moreover, with path-based timing constraint, the gates that are in paths with positive and negative slacks can be handled efficiently. This is not the case with gate-based timing constraints. On top of all these benefits, there is no need to do gate slack allocation, which is a hard problem as already stated in the introduction.

3.1 Single path optimization example

We first explain the parameters that are involved in the optimization of a single timing path. A sample path is shown in Figure 1. The slack at the output is obtained from the sign-off tool at the path's endpoint. We assume this path has a positive slack, s_1. The leakage power consumptions for version k of gate1 and gate2 are p_{1k} and p_{2k}, respectively, which are available from cell library characterization data.

Definition 4: **Delay arc** for each gate is defined as the delay from each one of its inputs to its output.

Hence, there are different delay arcs for different inputs of a gate. In addition, there are two other delay arcs with each gate on some path of interest. They denote interconnection delays from output of the preceding gate to the input pin of the gate in question and from output of the gate in question to the input pin of the succeeding gate.

The optimization begins with an initial solution, which is the already sized circuit available after sign-off. As stated earlier, in this example, there is a positive slack which can be traded for leakage power minimization. Parameter d_{jk} is introduced which is the change in the delay of the three arcs (gate and two interconnects delay arcs) associated with gate j when it is swapped by version k. d_{jk}'s are constants obtained by using the **estimate_eco** command in our sign-off tool [9]. These constants can obviously be extracted in O($n.k$) time, where n is the number of the gates and k is the number of the versions for each gate. It is necessary to mention that in the d_{jk}'s calculation, the input slew and process variations are all taken into account.

3.2 Problem formulation

The problem formulation for multiple paths is the same as the single path. The only difference is that the d_{jk} parameter is

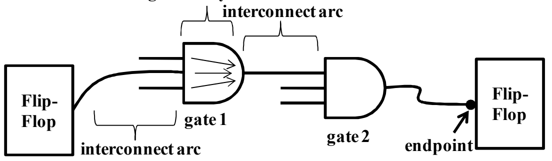


Figure 1. Example of single timing path

changed to d_{ijk} which is the change in the delay of gate j and two adjacent arcs in path i when it is swapped by version k. Figure 2 shows a gate which is in two different timing paths. Therefore,

$$\begin{cases} d_{13*} = \triangle\, arc2 + \triangle\, arc3 + \triangle\, arc4 \\ d_{23*} = \triangle\, arc1 + \triangle\, arc3' + \triangle\, arc5 \end{cases}$$

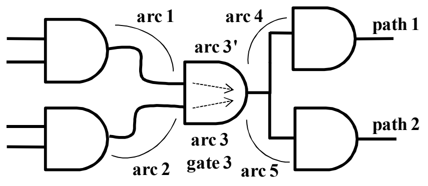


Figure 2. Gate in different paths

The path-based leakage minimization problem is formulated as follows:

$$minimize\ O = \sum_j \sum_k p_{jk} x_{jk} \tag{1}$$

$$s.t. \sum_{j \in path\ i} \sum_k d_{ijk} x_{jk} \leq S_i \quad \text{for } i=1,\dots,m \text{ number of paths} \tag{2}$$

$$\sum_k x_{jk} = 1 \qquad \text{for } j=1,\dots,n \tag{3}$$

$$x_{jk} = \begin{cases} 1 & if\ vesion\ k\ of\ gate\ j\ is\ chosen \\ 0 & otherwise \end{cases} \tag{4}$$

where p_{jk} is the leakage power consumption for version k of gate j. So, it is a positive constant and available from the cell library.

In the case that the path slack is negative, S_i is zero. In case of positive slack, it is equal to the slack value.

Note: We cannot include all paths of the circuit in our optimization problem since there are an exponential number of them. A **fixed number** of worst timing paths per circuit endpoint in the design are chosen to be involved in the optimization. This makes the number of paths in the optimization constant. These paths are obtained from the timing sign-off tool. This is indeed not an optimal approach; however, in practice, this proves to be a highly effective approach. If we get up to 100 (worst) paths per circuit endpoint, the chance that the sizing damages the 101st path ending at that same endpoint is quite small. Furthermore, in practice, most of these 100 worst timing paths per endpoint have positive slacks since the design is at the sign-off stage and the number of timing violations is small.

This problem formulation is for all endpoints at the same time meaning that all the paths chosen for each endpoint are timing constraints of one mathematical problem. The reason is that some gates may lie on multiple paths which end at different endpoints. It is also worth mentioning that only gates that lie on timing paths that are considered during the current optimization step are allowed to be re-versioned; i.e., these gates are the only ones included in the current optimization process. The effectiveness of our path pruning technique (choosing 100 worst timing paths per endpoint) is shown in section 6 when experimental results are presented.

3.2.1 Simultaneous changes in adjacent gates

There may be a situation where two adjacent gates are swapped simultaneously. Constraint (2) cannot take care of this issue and simply adds up the change in the interconnect delay between the two gates once for the re-versioning of the first gate and again for swapping the second gate. Therefore, constraint (2) is modified to fix this problem. Figure 3 illustrates the circuit model for two adjacent gates. R_i and R'_i denote the driving resistances of old and new versions of gate i, respectively. C_j and C'_j are the input capacitances of old and new versions of gate j, respectively. R_{net}

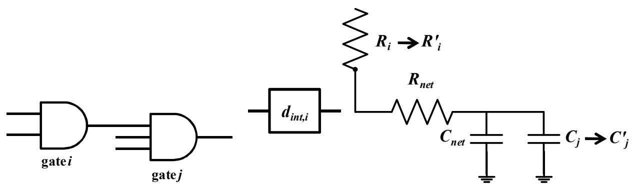


Figure 3. Simultaneous changes in adjacent gates

and C_{net} are the lumped resistance and capacitance of the interconnect between gates i and j. $d_{int,i}$ is the intrinsic delay of gate i.

The error in the model, which occurs when two adjacent gates change simultaneously is $(R'_i - R_i)(C'_j - C_j)$. The derivation is omitted for brevity. Hence, constraint (2) is modified to:

$$D_i = \sum_{j\in path\, i} \sum_k d_{ijk} x_{jk} + \sum_{j_1\in path\, i} \sum_{j_2\in path\, i} \sum_{k_1} \sum_{k_2} m_{j_1 j_2 k_1 k_2} x_{j_1 k_1} x_{j_2 k_2} \leq S_i \quad (5)$$

where

$$m_{j_1 j_2 k_1 k_2} = \begin{cases} -(R'_i - R_i)(C'_j - C_j) \ \ if\ j_1\ and\ j_2\ are \\ immediate\ neighbors \\ 0\ otherwise \end{cases} \quad (6)$$

$$x_{j_1 k_1} = \begin{cases} 1 & if\ \ new\ version\ of\ gate\ j_1 is\ used \\ & 0\ otherwise \end{cases} \quad (7)$$

$$x_{j_2 k_2} = \begin{cases} 1 & if\ new\ version\ of\ gate\ j_2 is\ used \\ & 0\ otherwise \end{cases} \quad (8)$$

4. SOLUTION METHOD

Conjugate gradient method is chosen to solve the optimization problem [10]. This method is an algorithm for the numerical solution of a system of linear or nonlinear equations. It is an iterative method, so it can be applied to sparse systems, which are too large to be handled by direct methods.

Conjugate gradient method has been used in other steps of EDA tools such as placement [11][12] and area minimization of power distribution network [13].

4.1 Conjugate Gradient

The nonlinear conjugate method is designed to solve unconstrained optimization problems [10]:

$$minimize\ f(x)$$

Here $f: R^n \rightarrow R$ is a ***smooth***, nonlinear function whose gradient is denoted by g.

Definition 5: Functions that have continuous derivatives up to some desired order over some domain are called ***smooth***.

In general, the conjugate gradient method finds the minimum by executing a series of line minimizations. In the k^{th} iteration, the conjugate direction (p_k) is computed based on the current gradient (g_k) and the gradient from the last iteration (g_{k-1}). Then the conjugate direction and the result of the line minimization (α_k) are used to find the next solution point (x_{k+1}).

$$p_k = -g_k + \beta_k p_{k-1}$$
$$x_{k+1} = x_k + \alpha_k p_k$$

There are different expressions for β_k which are basically the only difference between various conjugate gradient methods. One of the best methods is proposed by Polak-Ribiere, where,

$$\beta_k = \frac{g_k^T (g_k - g_{k-1})}{g_{k-1}^T g_{k-1}}$$

4.2 Formulation of the Penalty Function

Since the conjugate gradient method is for the optimization of unconstrained problems, we transform our constrained optimization problem to the unconstrained one by using the penalty method [14]. The basic idea of the penalty method is to add new terms to the objective function that consists of a penalty parameter and a measure of violation of the constraints. The measure of violation is nonzero when constraints are violated and is zero in the region where constraints are met.

The objective function to be solved by the conjugate gradient method must be a smooth function [10]. Hence, the constraints in our formulation should be changed to a new set of constraints that are smooth and greatly increase the objective function value when they are violated.

In the remainder of this section, we will introduce and apply a number of transformations using smooth mathematical functions in order to manipulate constraints of the gate version selection problem so that the problem can be solved in a numerically efficient manner with the CG and penalty methods. To the best of our knowledge, these transformations have not previously been presented in the solution of any EDA problem.

The new objective function would be in the form of:

$$O' = O + Pt \quad (9)$$

$$Pt = c \cdot \sum all\ the\ transformed\ constraints \quad (10)$$

where Pt is the penalty function and $c = 10(\sum_j p_{j1} x_{j1})$ is the cost parameter. c is 10 times the maximum possible leakage power consumption, assuming k=1 corresponds to the highest leakage version for each gate.

The inequality constraint (5) is manipulated by using *tangent hyperbolic* function. The value of tangent hyperbolic function changes between -1 and 1.

Theorem: *Tangent hyperbolic* is a smooth function.

Proof: $tanhx = \frac{e^{2x}-1}{e^{2x}+1}$ and $\frac{d}{dx} tanhx = \frac{2e^{2x}(e^{2x}+1) - 2e^{2x}(e^{2x}-1)}{(e^{2x}+1)^2}$

There is $(e^{2x}+1)^{n+1}$ in the denominator of derivatives of order n which is always greater than 1. Hence, tangent hyperbolic has derivatives of all orders and is a smooth function. ■

We shift and scale tangent hyperbolic to make the cost zero for delay changes less than S_i and infinity (i.e., 10 times more than the minimum possible leakage power consumption) for the delay changes more than S_i. The new constraint which is incorporated in the objective function through the penalty method is as follows (it is also depicted in Figure 4(a))

$$D'_i = [1 + \tanh(\alpha \cdot (D_i - S_i))] \quad \text{for } i=1,\dots,m \quad (11)$$

where α is a smoothing parameter.

Smoothing factor is low during the first iterations of the conjugate gradient solver but its value is increased as we get closer to the final solution. This adaptive smoothing parameter makes the run time of the solver per iteration shorter at the beginning (although it also results in lower accuracy of the solution) but increases the run time per iteration toward the end of the optimization process (while improving the solution's accuracy.)

The equality constraint (3) is first changed to two inequality constraints $\sum_k x_{jk} \leq 1$ and $\sum_k x_{jk} \geq 1$, and then transformed by using *tangent hyperbolic* with smoothing factor β and shift value μ which changes around 0 and is decreased as β is increased. As illustrated in Figure 4(b) the function is 0 when $\sum_k x_{jk} = 1$ and high when the cost parameter is multiplied in it. The figure also depicts the transformed function with two different μ values. The transformed function is as follows:

$$\left[1+\tanh\left(\beta\left(\left(\sum_k x_{jk}\right)-1-\mu\right)\right)\right] - \left[-1+\tanh\left(\beta\left(\left(\sum_k x_{jk}\right)-1+\mu\right)\right)\right] \quad \forall j \tag{12}$$

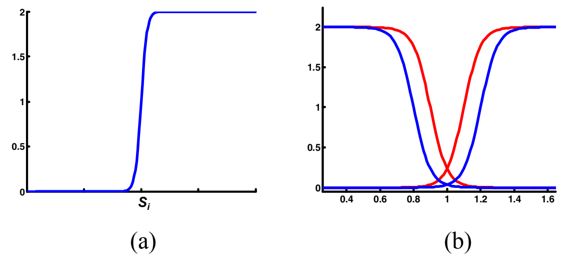


Figure 4. (a) Constraint (11), (b) Constraint (12), μ=0.1 and 0.2 for the blue and red lines, respectively.

The last constraints to be modified are (4), (7) and (8). Since their transformed functions are all the same type, we just explain it for constraint (4).

This cost function needs to be zero around 0 and 1, large elsewhere and indeed a smooth function. We choose *Chebyshev polynomial* since it has these features.

Definition 6: *Chebyshev polynomial* of degree i is defined as

$$T_i(x) = \frac{1}{2}\left[\left(x+\sqrt{x^2-1}\right)^i + \left(x-\sqrt{x^2-1}\right)^i\right] \tag{13}$$

The chebyshev polynomials have the property that $|T_i(x)| \leq 1$ on the domain $x \in [-1,1]$, and furthermore that $|T_i(x)|$ is maximum on the domain $x \notin [-1,1]$ among such polynomials.

Theorem: Chebyshev polynomial is smooth.

Proof: Chebyshev polynomial has derivatives of all orders. It might seem that it is not differentiable at $x = \pm 1$ but it is indeed differentiable and the derivatives are as follows:

$$\left.\frac{d^p T_n}{dx}\right|_{x=\pm 1} = (\pm 1)^{n+p} \prod_{k=0}^{p-1} \frac{n^2-k^2}{2k+1}. \quad \blacksquare$$

Chebychev polynomial of degree 4 is chosen as the cost function for constraint (4) since it has two minima. It should be manipulated by shifting and compression to become zero at 0 and 1. The cost function is as follows:

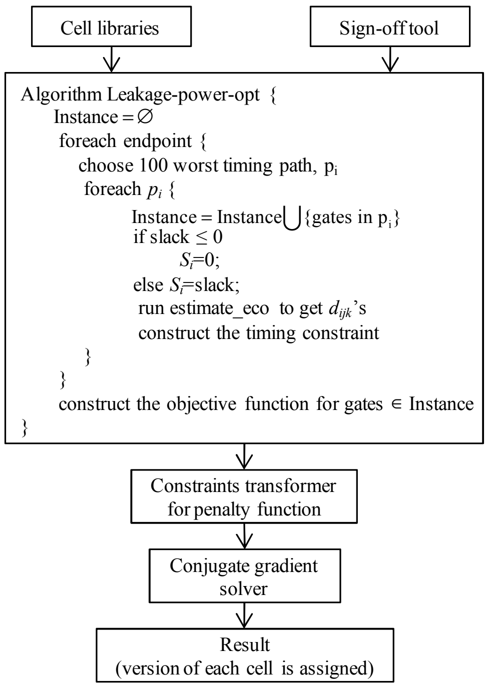
$$\left[2+\left((1.4\cdot x_{jk}-0.7)+\sqrt{(1.4\cdot x_{jk}-0.7)^2-1}\right)^4 + \left((1.4\cdot x_{jk}-0.7)-\sqrt{(1.4\cdot x_{jk}-0.7)^2-1}\right)^4\right]^\gamma \quad \forall j,k \tag{14}$$

where γ is a smoothing parameter between 0 and 1. In the beginning γ is 1 and decreases by each iteration. Notice that there is a look up table to pick up values of the smoothing parameters during various iterations of the conjugate gradient method.

5. OPTIMIZATION FLOW

Figure 5 summarizes the flow of our optimization algorithm. It also includes the pseudo code for construction of the problem formulation.

The information about d_{ijk}, timing paths, slack values, and connectivity of the gates in the circuit passes from sign-off tool to the problem formulation constructor. p_{jk}'s are obtained from cell libraries. In the next step, constraints of the problem formulation are transformed and added to the objective function using penalty terms. Finally, the conjugate gradient solver gives the results, which are basically the assigned version of each gate in the design.



Cell libraries
Sign-off tool
Constraints transformer for penalty function
Conjugate gradient solver
Result (version of each cell is assigned)

```
Algorithm Leakage-power-opt {
  Instance = $\varnothing$
  foreach endpoint {
    choose 100 worst timing path, $p_i$
    foreach $p_i$ {
        Instance = Instance $\bigcup$ {gates in $p_i$}
        if slack $\leq 0$
            $S_i$=0;
        else $S_i$=slack;
        run estimate_eco to get $d_{ijk}$'s
        construct the timing constraint
    }
  }
  construct the objective function for gates $\in$ Instance
}
```

Figure 5. Optimization flow

6. EXPERIMENTAL RESULTS

Our post sign-off leakage power optimization algorithm was applied to the circuits that have already gone through the complete CAD flow including synthesis using dual-V_{th} framework; i.e., the test circuits had already been optimized to use dual-V_{th} logic gates before our new optimization was applied to them. We emphasize that our post sign-off optimization algorithm does not change the number of threshold voltage levels which are available for leakage power minimization. Moreover, these versions of each logic gate were already available during standard synthesis flow and no new gate versions were added to the cell libraries for the purpose of our optimization algorithm. So, in summary, all leakage reduction percentages reported describe the additional decrease achieved by removing the remaining slack from a dual-V_{th} circuit without adding any new gates to the cell library.

The proposed optimization problem using conjugate gradient method has been implemented in *C*. We tested our program with real industry's test cases. These test cases have complexities ranging from 16,116 to 84,928 numbers of instances and 16,430 to 73,407 numbers of nets. Table 1 shows the specification of our test cases.

The objective of the optimization problem is to minimize the leakage power considering timing constraints. This means the number of timing violations of the circuit is not allowed to increase after the optimization. The timing criteria of a circuit can be considered with different parameters. Some of these criteria, which are commonly used in the technical discussions, are as follows:

Definition 7: **WNS** is defined as the worst negative slack among all the timing paths of a circuit.

Table 1: Specification of test cases

Test case	Number of instances	Number of nets	Technology node	Number of V_{th}
A	64806	65150	90nm	2
B	16116	18142	65nm	3
C	70208	64284	65nm	2
D	84928	73407	65nm	2
E	74655	69590	65nm	2
F	21754	16430	65nm	2
G	21929	16620	65nm	2

Definition 8: **TNS** is the summation of the worst negative slacks of all endpoints in a circuit.

Table 2 shows the optimization results for each test case separately. These results are in comparison with no post sign-off leakage power optimization. As the result of the post sign-off optimization, the runtime is increased significantly which is reasonable. Although this increase should not be huge since turn-around time is an important factor in EDA industry.

WNS and TNS for each case are mentioned to show how the worst negative slacks are changed after the optimization. Setup and hold times are also considered as critical timing characteristics in a circuit. The change in the total number of setup and hold times violations is a good metric to represent the efficiency of our path pruning technique. The path pruning technique considers only the worst 100 timing paths per endpoint as the timing constraints of the optimization formulation.

There are two reasons for decrease in total number of setup and hold times violations in some of the test cases. One reason is reduction in the number of hold violations since sizing down the gates (or using bigger channel length) in a path (to minimize leakage power) can slow down the path and remove the hold violation if exists. The reason for reduction in setup (or slack) violations is that the solution space is not continuous. The timing constraint only says that the new slack in violating paths should be greater than or equal to the old slack. So, it is certainly possible to find a solution that gives a better slack if an exact one cannot be found.

The average results for all the test cases together in comparison with sign-off without any power optimization are shown in Table 3. The average leakage power by doing post sign-off optimization is decreased by 34%. On the other hand, the runtime is increased by 77% which is reasonable. Furthermore, the increase in the number of timing violations is negligible. The average of total number of setup and hold times violations is increased by just 0.9%, which again proves our claim for effectiveness of our path pruning technique.

Table 2: Optimization results for different test cases

Test case	WNS (%)	TNS (%)	Total # of setup/hold times violations (%)	Runtime (%)	Leakage power (%)
A	0	-1.4	+1.5	+66	-44.6
B	-0.1	0	0	+91	-30.7
C	0	+4.9	-3.1	+71	-27.3
D	-8.4	-22	0	+97	-29.2
E	-0.9	-1.3	-72	+60	-21.7
F	-1.9	-11.6	+73.5	+62	-39.3
G	-1.1	+51.8	+6.7	+95	-48.4

Table 3: Average result of optimization

WNS (%)	TNS (%)	Total # of setup/ hold times violations (%)	Runtime (%)	Leakage power (%)
-1.7	+2.9	+0.9	+77	-34.4

7. CONCLUSION

In this paper, we took advantage of positive slacks that become available using accurate sign-off tools. These slacks were traded for leakage power minimization. We introduced our optimization problem and used path-based timing constraints. We showed that by using the path-based timing constraints with the help of the information obtained from the sign-off tool, we could setup a problem formulation which was more accurate than previous gate-based ones. We proposed a path pruning technique to reduce the number of timing constraints. Conjugate gradient method was used to solve the optimization problem. The constraints were transformed to penalty functions by means of our novel technique and added to the objective function. Finally, experimental results on real industrial test cases confirmed the strength and efficacy of our optimization algorithm.

REFERENCES

[1] International technology roadmap for semiconductors, 2009, http://www.itrs.net/

[2] P. Pant, R. K. Roy, and A. Chatterjee, "Dual-Threshold Voltage Assignment with Transistor Sizing for Low Power CMOS Circuits," *IEEE Trans.on VLSI Systems*, 2001.

[3] S. Sirichotiyakul et al., "Stand-by power minimization through simultaneous threshold voltage selection and circuit sizing," *Design Automation Conference*, 1999.

[4] M. Manil et al., "An Efficient Algorithm for Statistical Minimization of Total Power under Timing Yield Constraints," *Design Automation Conference*, 2005.

[5] X. Ye, Y. Zhan, and P. Li, "Statistical Leakage Power Minimization Using Fast Equi-Slack Shell Based Optimization," *Design Automation Conference*, 2007.

[6] K. Jeong, A. B. Kahng, and H. Yao, "Revisiting the Linear Programming Framework for Leakage Power vs. Performance Optimization," *Int'l Symposium on Quality of Electronic Design,* 2009.

[7] X. Yang, B. Choi, and M. Sarafzadeh, "Timing-Driven placement using Hierarchy Guided Constraint Generation," *Int'l Conference on Computer Aided Design*, 2002.

[8] N.V. Arvind et al., "Path Based Approach for Crosstalk Delay Analysis," *VLSI Design Conference,* 2004.

[9] Synopsys PrimeTime. http://www.synopsys.com/Tools/ Implementation/SignOff/ Pages/PrimeTime.aspx

[10] L. Adams and J. L. Nazareth, "Linear and Nonlinear Conjugate Gradient-Related Methods," *SIAM*, 1996.

[11] W. Naylor et al., "Non-Linear Optimization System and Method for Wire Length and Delay Optimization for an Automatic Electric Circuit Placer," U.S. Patent 6301693, Oct. 2001.

[12] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer," *IEEE Trans. on Computer Aided Design*, 2005.

[13] X. Wu et al., "Area Minimization of Power Distribution Network Using Efficient Nonlinear Programming Techniques," *IEEE Trans. on Computer Aided Design*, 2004.

[14] M. Krizek et al., "Conjugate Gradient Algorithms and Finite Element Methods," *Springer*, 2004.