A Fast Simultaneous Input Vector Generation and Gate Replacement Algorithm for Leakage Power Reduction

LEI CHENG, DEMING CHEN, and MARTIN D. F. WONG
University of Illinois at Urbana-Champaign

The Input vector control (IVC) technique is based on the observation that the leakage current in a CMOS logic gate depends on gate input state, and a good input vector is able to minimize leakage when the circuit is in sleep mode. The gate replacement technique is a very effective method to further reduce the leakage current. In this article, we propose a fast heuristic algorithm to find a low-leakage input vector with simultaneous gate replacement. Results on MCNC91 benchmark circuits show that our algorithm produces 14% better leakage current reduction with several orders of magnitude speedup in runtime for large circuits compared to the previous state-of-the-art algorithm. In particular, the average runtime for the ten largest combinational circuits has been dramatically reduced from 1879 seconds to 0.34 seconds.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided design (CAD)*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Input vector control, leakage reduction, gate replacement

ACM Reference Format:

Cheng, L., Chen, D., and Wong, M. D. F. 2008. A fast simultaneous input vector generation and gate replacement algorithm for leakage power reduction. ACM Trans. Des. Autom. Electron. Syst., 13, 2, Article 34 (April 2008), 15 pages, DOI = 10.1145/1344418.1344430 http://doi.acm.org/10.1145/1344418.1344430

1. INTRODUCTION

CMOS technology scaling is the main force that drives the semiconductor industry. Together with the power-supply voltage scaling, it renders chip

A preliminary version of this article was presented in *the IEEE/ACM Design Automation Conference (DAC)* in July 2006.

Authors' addresses: L. Cheng, Department of Computer Science/Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801; email: lcheng1@uiuc.edu; D. Chen, M. D. F. Wong, Department of Electrical and Computer Engineering/Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

performance substantially improved while power consumption remains reasonable. However, to maintain high performance with lower supply voltage, transistor threshold voltage also needs to be scaled down. Lowering the threshold voltage exponentially increases the leakage current of the device. It is reported that 25% total energy is dissipated by leakage current in a 90 nm Itanium processor core [Naffziger et al. 2005].

Leakage reduction techniques have been proposed previously on device, circuit, and system levels. Dual threshold voltage (V_t) transistors [Wei et al. 1998] and body biasing [Kobayashi and Sakurai 1994] techniques change threshold voltage, and effectively reduce leakage. However, these techniques need process changes. Meanwhile, stack transistors can effectively reduce leakage. A two-transistor stack can reduce leakage by an order of magnitude as compared to a single transistor [Ye et al. 1998]. Based on this theory, sleeping transistor Johnson et al. 1999a and gated-V_{dd} [Mutoh et al. 1995] techniques were introduced to reduce the leakage power.

Input vector control (IVC) is another way to efficiently reduce leakage power [Halter and Najm 1997; Ye et al. 1998; Rao et al. 2003]. Given a circuit, an optimal input vector exists to minimize the leakage current for this circuit. However, finding such an optimal input vector is NP-hard [Johnson et al. 1999b]. For large circuits with deep levels, IVC becomes less effective because the controllability of internal nodes is lower. A gate replacement technique can be used to improve the controllability. It can also be used to further reduce leakage.

Most of the proposed heuristics for IVC become slow for large circuits. The fastest implementation, in Rao et al. [2003], is $O(n^2)$, where n is the number of gates in the circuit. Considering and gate replacement, the computation complexity further increases. Existing methods for gate replacement are formulated assuming the input vector is known, and that gate replacement is carried out after input vector generation. However, these two techniques energetically interact. Without consideration of their interaction, the leakage reduction results are obviously not optimal.

In this article, we present an $O(n)$ heuristic algorithm to solve the IVC and gate replacement problems simultaneously. A dynamic-programming-based algorithm is used for making a fast evaluation of input vectors, as well as for replacing gates. Our contributions can be summarized as follows.

—We propose a link-deletion-based decomposition method for circuits which is much more effective for leakage reduction than previous tree decomposition algorithms.
—We propose a linear-time dynamic programming algorithm that produces the optimum input vector with simultaneous gate replacement for tree circuits.
—We propose an iterative method that can very quickly produce the low-leakage input vector and gate replacement solution for a circuit.
—Our algorithm is 11 times faster than the pervious fastest algorithm for input vector generation [Rao et al. 2003], on average; in that paper gate replacement was not considered.

Table I. Leakage Current of a NAND2 Gate

INPUT	Leakage (nA)
00	37.84
01	100.30
10	95.17
11	454.50

Data obtained by simulation in Cadence Spectre using 0.18 μm process.

—Our algorithm produces better results than the previous state-of-the-art IVC algorithm with gate replacement [Yuan and Qu 2005] with several orders of magnitude speedup in runtime.

In Section 2, we will briefly review existing methods to solve the IVC problem and the current approach for gate replacement. Section 3 presents our fast leakage reduction heuristic with simultaneous gate replacement. The results are shown in Section 4, and we will conclude this article in Section 5.

2. PRELIMINARIES AND RELATED WORK

2.1 Input Vector Control

Input vector control (IVC) methods were first proposed in Halter and Najm [1997] and Ye et al. [1998]. This technique is based on the observation that the leakage current in a CMOS logic gate is dependent on the gate input state (see Table I of Yuan and Qu [2005]). However, given a circuit, it is NP-hard to find a vector that minimizes the leakage [Johnson et al. 1999b]. Both a SAT-based algorithm [Aloul et al. 2002] and an integer-linear-programming-based algorithm [Gao and Hayes 2004] were proposed previously to find exact solutions. A faster algorithm for finding exact solutions was presented in Chopra and Vrudhula [2006, 2004]. In Chopra and Vrudhula [ibid.] a balanced min-cut algorithm was used to partition the circuits, then pseudo-Boolean enumeration was used to find input vectors for small blocks. However, these techniques for finding exact solutions are not practical for large circuits. A random search algorithm was proposed by Halter and Najm [1997], in which randomly chosen vectors were applied to the circuit and that vector which gave the least observed leakage value was reported. It was reported that a random search over 10000 vectors gave over 99% confidence that less than 0.5% of the vector population would have leakage lower than the minimum leakage value observed from the random search [Halter and Najm 1997; Rao et al. 2003]. An $O(n^2)$ algorithm was proposed in Rao et al. [2003]. Based on the controllability idea from circuit testing, that paper was able to get a result very close to that of an extensive random search method, with much lower computational cost compared to other proposed methods.

In Yuan and Qu [2005], a genetic algorithm was proposed to find the input vector with gate replacement. This algorithm is carried out in the following steps. Firstly, the circuit is decomposed into tree circuits, secondly, a dynamic

Fig. 1. Gate replacement example.

programming algorithm is applied to every tree circuit, followed by a heuristic gate replacement algorithm; finally, the tree circuits are connected by a genetic algorithm. The tree decomposition method used in this algorithm decomposes the circuit into a large number of small trees, which makes the dynamic programming algorithm less effective as compared to the genetic algorithm. This algorithm requires a long runtime for large circuits due to the nature of the genetic algorithm. For example, its average runtime on the ten largest combinational MCNC91 circuits is 1879 seconds. These ten circuits are not actually very large, and the largest has only 4000 gates. So the genetic algorithm is not scalable to large circuits with multiple millions of gates.

2.2 Gate Replacement

When paths in circuits become deeper, IVC techniques become less effective because gates with deep levels are harder to affect with the input vector. One way to solve this problem is the internal signal-control method proposed in [Abdollahi et al. 2004]. With a small delay overhead, it can reduce more leakage current. This method modifies gates using the stack transistor idea. It was assumed that every gate could be modifies to insert a control. Together with its SAT-based algorithm for input vector generation, the method could be very expensive for solving the problem for a large circuit.

The internal signal-control method needs to modify gates. On the other hand, Yuan and Qu [2005] reduced gate leakage by gate replacement only using gates available in the library. In Yuan and Qu [ibid.], an optimum input vector is first produced by a dynamic programming algorithm, followed by a heuristic gate replacement algorithm. However, this separation of input vector generation and gate replacement is not able to produce an optimum solution because the output of a gate may be changed after replacement (in Figure 1, G^* outputs 1 when $Sleep = 1$), which changes the leakage currents of gates to which it fans out. In this article, these two methods are considered simultaneously through a dynamic-programming-based technique. The advantage of our method is that we are able to find the minimum-leakage input vector for tree circuits, with simultaneous gate replacement. For general circuits, we are able to achieve better leakage reduction results as compared to the separate approach used in Yuan and Qu [2005].

The gate replacement technique replaces one gate $G(\vec{x})$ by another $\tilde{G}(\vec{x}, Sleep)$, where $\vec{x}$ is the input vector at G such that the following holds [Yuan and Qu 2005].

(1) $\tilde{G}(\vec{x}, 0) = G(\vec{x})$ when the circuit is active ($Sleep = 0$).
(2) $\tilde{G}(\vec{x}, 1)$ has less leakage than $G(\vec{x})$ when the circuit is in standby mode ($Sleep = 1$).

Algorithm 1. Overall Algorithm

Input $\{G_1, \ldots, G_n\}$: a circuit with n gates;
N_i : the number of iterations to run;
Output: $\vec{V}_{opt}$: an input vector producing small leakage;

```
Convert the circuit into trees;
Assign initial values to dangling inputs;
for i = 1 to N_i do
    Perform dynamic programming algorithm for each tree;
    Adjust the dangling assignment;
    CheckOscillations;
    if solutions converge then
        break;
end
```

Figure 1 shows how to replace a NAND2 gate. According to the leakage data (see Table I), the leakage of NAND2 with the vector 11 is $454.50nA$, and it is reduced to $94.87nA$ after the NAND2 gate is replaced by a NAND3.

3. SIMULTANEOUS INPUT VECTOR GENERATION WITH GATE REPLACEMENT

In this section, we present our algorithm for simultaneous input vector generation with gate replacement. Several key techniques are used, such as link deletion, dangling input assignment, dynamic programming, and oscillation checking. Algorithm 1 is a global overview of our algorithm. In the beginning, the circuit is transformed into large trees. This transformation is made by deleting connections among gates until every gate fans out to at most one other gate. Removing connections in a circuit obviously creates many dangling inputs (a dangling input of a gate is the nonprimary input which has no fanin gate). In the second step, we assign initial values to these dangling inputs.[1] The value of a dangling input is always equal to the estimated output value of its fanin gate before deleting the connections. For example, if the estimated output of gate G_1 is 1 as in Figure 2(b), the value of the dangling input of G_3 is 1. Then we iteratively perform the dynamic programming algorithm on the trees we created. Each iteration computes the best input vector with regard to the current dangling assignment. The best input vector of the current iteration is used to update the dangling assignment for the next iteration. During each iteration, we check whether the input vectors computed by different iterations oscillate, and resolve this problem if it happens. When two consecutive iterations produce the same best input vector, the algorithm converges and terminates. If the algorithm does not converge, it will terminate after N_i iterations. We find that $N_i = 20$ is good enough for practical problems. Details will be presented in the following subsections.

[1]Hereafter, a value assignment of all dangling inputs is abbreviated as a dangling assignment.

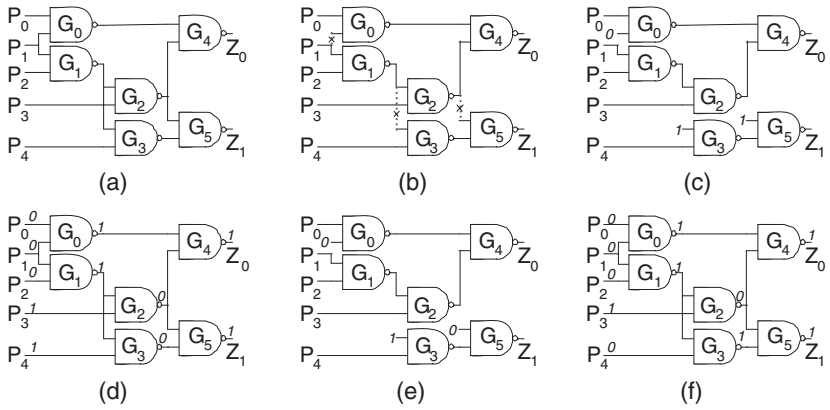

Fig. 2. Step-by-step illustration of our algorithm on the circuit $C17$: (a) the original $C17$ circuit; (b) tree circuits after deleting some connections between P_1 and G_0, G_1 and G_3, and G_2 and G_5; (c) we assign values 0, 1, and 1 to dangling inputs of gates G_0, G_3, and G_5, respectively; (d) the current optimum input vector (0, 0, 0, 1, 1) and corresponding gate outputs; (e) the dangling assignments have been changed to 0, 1, and 0 for gates, G_0, G_3, and G_5, respectively; (f) the new best input vector (0, 0, 0, 1, 0), which is the optimum input vector of circuit $C17$.

3.1 An Example

Figure 2 provides an example of running our algorithm on a simple MCNC91 benchmark circuit $C17$ (Figure 2(a)). In the first step, we transform the circuit into trees by temporally deleting connections between P_1 and G_0, G_1 and G_3, and G_2 and G_5 (Figure 2(b)). After deletion, gates G_0, G_3, and G_5 have dangling inputs. In the second step, we assign values 0, 1, and 1 to the dangling inputs of gates G_0, G_3, and G_5, respectively (Figure 2(c)). Since the circuit has been transformed into trees, we can apply a dynamic programming algorithm to determine its minimum leakage and the corresponding input vector (Figure 2(d)). With this input vector, we are able to evaluate all the gates and calculate their outputs. We use the outputs of gates to update values of dangling inputs for the next iteration. In Figure 2(e), the values of dangling inputs of gates G_0, G_3, and G_5 have been changed to 0, 1, and 0, respectively. With the new dangling assignment, we run the algorithm another time and get the new best input vector (Figure 2(f)). This new input vector produces the same dangling assignment as the previous iteration, and the algorithm converges and terminates. The input vector calculated by our algorithm in this example happens to be the optimum solution for the circuit $C17$.

3.2 Link Deletion

At the beginning of our algorithm, we transform the given circuit into trees by deleting connections between gates so that every gate fans out to at most one other gate. If a gate G fans out to k other gates $G_1 \ldots G_k$, we pick a best gate G_i according to some heuristic rules, and delete connections between all the other $k-1$ gates and G. The heuristic rules determining whether G_i is better than G_j are presented as follows.

—If the longest path between G_i and one primary output (denoted as $LP(i)$) is longer than that of G_j, G_i is better than G_j. The intuition behind this rule is that the longer the path, the larger the number of gates affected by this gate, and it is more appropriate to keep intact the connections between this gate and its inputs.

—If $LP(i) = LP(j)$ and gate G_i fans out to more gates than does G_j, then G_i is better than G_j.

—Otherwise, we make a random decision.

In Figure 2(a), G_1 fans out to both G_2 and G_3. We can see that $LP(2) = LP(3) = 2$, but G_2 has two fanouts (G_4 and G_5) while G_3 has only one, so we delete the connection between G_1 and G_3 (Figure 2(b)).

Note that our link-deletion-based transformation is different from the tree decomposition method in Yuan and Qu [2005], in which every gate with multiple fanouts is the root of a decomposed tree. Their method produces a large number of trees, and the size of each tree is very small. In our algorithm, only the primary outputs of the circuit are the roots of trees. A small number of gates in a tree makes the dynamic programming algorithm on trees very ineffective, because the interactions among trees are more important in this case. In Yuan and Qu [2005], the average number of gates in a tree for the benchmark circuits is 6, while our number is 72.

3.3 Initial Dangling Assignment

In our algorithm, the value of a dangling input is always equal to the output of the corresponding fanin gate in the original circuit. We design three heuristics to estimate the gate outputs at the beginning of our algorithm.

—*Random Assignment*. We randomly assign a value as the output of a gate.

—*Probability Estimation*. In this method, we assume that the occurrence of every input combination of a gate has the same probability. With this assumption, we are able to calculate the probabilities of the different output values of a gate. For example, in Figure 2, the gate G_1 is a NAND gate, and the probability of producing 1 is 0.75, which is larger than that of producing the value 0. So, the estimated output of G_1 is 1.

—*Minimum-Leakage Estimation*. In this method, we apply the dynamic programming algorithm to the unconverted DAG circuit. The difference here is that a gate G may fan out to multiple gates $G_1 \ldots G_k$, and these gates may require different input values from G. In this case, we count the number n_1 of gates requiring the value 1, and the number n_0 of gates requiring the value 0, then compare n_1 with n_0. If $n_1 > n_0$, the output of G is set to the value 1. If $n_1 < n_0$, the output of G is set to the value 0. If there is a tie, we make a random assignment.

Our experiments indicate that the last method is superior. So we present our experimental results using the third heuristic in this article.

Table II. Local Variables in Algorithm 2

Variable	Definition
$N(i)$	the number of inputs of gate G_i
$I(i, j)$	the jth input of gate G_i
$Out(i, \vec{x})$	the output of gate G_i with its local input vector $\vec{x}$
$Out_R(i, \vec{x})$	the output of gate G_i with its local input vector $\vec{x}$ when G_i is replaced
$L(i, \vec{x})$	the leakage current of gate G_i with its local input vector $\vec{x}$
$L_R(i, \vec{x})$	the leakage of the replaced gate of G_i with local input vector $\vec{x}$
$Rep(i, z)$	indicate whether to replace gate G_i when its output is z
$LK(i, z)$	minimum total leakage of the subtree rooted at gate G_i when its output is z
$\vec{V}(i, z)$	the input vector producing $LK(i, z)$
$\vec{x}_j$	the value of the jth bit of vector $\vec{x}$

3.4 Optimum Algorithm for Tree Circuits with Simultaneous Gate Replacement

Given a tree circuit, the dynamic programming algorithm is able to find the input vector that produces minimum leakage. In the algorithm, the gates are sorted topologically, and evaluated one-by-one according to this order. Whenever we evaluate a gate G, we consider all combinations of its input values and find the minimum leakage $LK(G, 0)$ of the subtree rooted at the gate G when G's output is 0, the minimum leakage $LK(G, 1)$ of the same subtree when G's output is 1, and the corresponding optimum input vectors. After evaluating all gates, we are able to find the minimum leakage of the tree and the optimum input vector. This part of the algorithm is the same as that used in Yuan and Qu [2005].

We extend the algorithm so that it produces the optimum input vector for a tree circuit when we also carry out simultaneous gate replacement. The details of our algorithm are presented in Algorithm 2. Table II explains local variables used in our algorithm. In line 8, LK is the minimum total leakage of subtrees rooted at the inputs of gate G_i, and $LK(I(i, j), \vec{x}_j)$ is the minimum leakage of the subtree rooted at the jth input of gate G_i; in line 12, the input vector producing minimum leakage for the subtree rooted at gate G_i is generated by combining theinput vectors of G_i's fanin trees. In the algorithm, thevariable $valveRep$ is used to control the area and delay of the final circuit. Replacing gates increases the area and delay because one gate is always replaced by another gate with more transistors. The larger the variable $valveRep$, the smaller the number of gates replaced, thus the smaller the area and delay overhead due to gate replacement. When $doRep = 0$, the algorithm is able to produce a minimum-leakage vector without considering gate replacement; when line 14 is removed, our algorithm produces the minimum-leakage vector for a tree circuit with gate replacement. Details of the optimality are provided by the following theorems.

Algorithm 2. Algorithm Producing Optimum Input Vector with Gate Replacement for a Tree Circuit

Input: $\{G_1, \ldots, G_n\}$: gates in the tree circuit sorted topologically
$doRep$: replace gate when $doRep = 1$
$valveRep$: control gate replacement
Output: $\vec{V}_{opt}$: optimum input vector
$\vec{R}$: replace gate G_i if $\vec{R}_i = 1$

Local: $N(i)$, $I(i, j)$, $Out(i, \vec{x})$, $Out_R(i, \vec{x})$, $L(i, \vec{x})$, $L_R(i, \vec{x})$, $Rep(i, z)$, $LK(i, z)$, $\vec{V}(i, z)$ (see Table II)

```
1  begin
2    for i = 1 to n do
3      if G_i is a primary input then
4        LK(i, 0) = 0; V(i, 0) = 0;
5        LK(i, 1) = 0 ; V(i, 1) = 1;
6        continue;
7      for each valid input combination x of G_i do
8        LK = sum_{j=1}^{N(i)} LK(I(i, j), x_j);
9        z = Out(i, x);
10       if L(i, x) + LK < LK(i, z) then
11         LK(i, z) = L(i, x) + LK;
12         V(i, z) = U_{j=1}^{N(i)} V(I(i, j), x_j);
13         Rep(i, z)=No;
14       if doRep and L_R(i, x) × valveRep < L(i, x) then
15         z = Out_R(i, x);
16         if L_R(i, x) + LK < LK(i, z) then
17           LK(i, z) = L_R(i, x) + LK;
18           V(i, z) = U_{j=1}^{N(i)} V(I(i, j), x_j);
19           Rep(i, z)=Yes;
20     end
21   end
22   if LK(n, 0) > LK(n, 1) then
23     V_opt = V(n, 0);
24   else
25     V_opt = V(n, 1);
26   end
27   Calculate R in reverse topological order;
28 end
```

THEOREM 3.1. *For any gate G_i ($1 \le i \le n$) and $z \in \{0, 1\}$, let N_i denote the number of inputs of G_i, and G_{g_j} ($1 \le j \le N_i$) denote the jth fanin gate of G_i. If we do not consider gate replacement, there must exist an input vector $\vec{x}$ of G_i such that G_i outputs z with input vector $\vec{x}$ and $LK(i, z) = L(i, \vec{x}) + \sum_{j=1}^{N_i} LK(g_j, \vec{x}_j)$, where $\vec{x}_j$ is the value of the jth bit of $\vec{x}$.*

PROOF. For gate G_i and its desired output value z, there must exist an input vector $\vec{x}$ such that the total leakage of the subtree rooted at G_i is minimized and G_i outputs z with input vector $\vec{x}$. The total leakage of the subtree rooted at G_i is the summation of the leakage of G_i itself and the total leakages of all subtrees rooted at fanin gates of G_i, thus it is obvious that $LK(i,z) = L(i,\vec{x}) + \sum_{j=1}^{N_i} LK(g_j, \vec{x}_j)$. □

With this theorem, we directly have the following corollary.

COROLLARY 3.2. *When gate replacement is ignored ($doRep = 0$), Algorithm* 2 *produces the minimum-leakage vector for a tree circuit.*

In the case of considering gate replacement, we have the following results which are similar to Theorem 1 and Corollary 1.

THEOREM 3.3. *For any gate G_i ($1 \le i \le n$) and $z \in \{0,1\}$, let N_i denote the number of inputs of G_i, and G_{g_j} ($1 \le j \le N_i$) denote the jth fanin gate of G_i. With simultaneous gate replacement, there must exist two input vectors $\vec{x}_1$ and $\vec{x}_2$ such that $LK(i,z) = min\{L(i,\vec{x}_1) + \sum_{j=1}^{N_i} LK(g_j, \vec{x}_{1j}), L_R(i,\vec{x}_2) + \sum_{j=1}^{N_i} LK(g_j, \vec{x}_{2j})\}$, G_i outputs z with input vector $\vec{x}_1$, and the replaced gate of G_i outputs z with input vector $\vec{x}_2$.*

PROOF. Following a similar approach as used in Theorem 1. □

COROLLARY 3.4. *Algorithm* 2 *produces the minimum-leakage vector for a tree circuit with simultaneous gate replacement when line* 14 *is removed.*

Since the maximum number of inputs of a gate in a circuit is constant, the number of combinations of input values of a gate is also constant. It is easy to know that our algorithm is linear in each iteration. In the algorithm, every gate uses a constant number of resources,[2] so the space complexity of the algorithm is also linear to the number of gates. In all, we have the following theorem, considering runtime and space complexity.

THEOREM 3.5. *The runtime of Algorithm* 2 *is $O(n)$, and the memory usage is also $O(n)$, where n is the number of gates in the circuit.*

PROOF. In this theorem, we assume that the maximum fanin number of a gate in the technology library is N_f, which is a constant. In Algorithm 2, the runtime of lines 8, 12, and 18 is $O(N_f)$, and all other lines between line 8 and 19 have unit runtime. The loop starting from line 7 has at most 2^{N_f} iterations, and the loop starting from line 2 has n iterations, so the total runtime is $O(N_f 2^{N_f} n)$. Since N_f is constant, the runtime of the algorithm is $O(n)$. It is obvious that the space complexity of the algorithm is $O(n)$. □

[2]Note that $\vec{V}(i,z)$ needs linear space, but exists in our algorithm for the purpose of simplifying the illustration. In our implementation, this vector is only constructed for the root gate at the end of the algorithm by backtracing the entire tree.

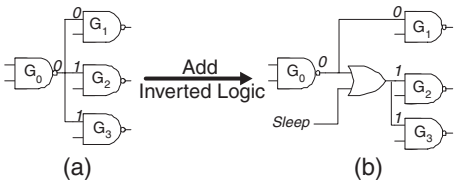

Fig. 3. Inverted logic insertion between G_0 and G_2, G_3.

3.5 Oscillation

Our algorithm converges and terminates after a few iterations for most cases. Occasionally, we observe oscillations. The oscillation happens when some input vectors produced by our algorithm repeat after several iterations. In our algorithm, each dangling assignment produces an input vector, and each input vector determines a new dangling assignment. The oscillation happens if and only if the dangling assignments repeat, so we check the oscillation by checking whether the dangling assignments in two nonadjacent iterations repeat.

Let $\vec{D}(i)$ denote the characteristic vector of the dangling assignment in iteration i of our algorithm, each bit of $\vec{D}(i)$ is the estimated output of a gate with multiple fanouts in iteration i. The oscillation happens if $\vec{D}(i) = \vec{D}(j)$ for some $i + 1 < j$ (if $i + 1 = j$, the algorithm converges). The equation $\vec{D}(i) = \vec{D}(j)$ can be checked efficiently by hashing each vector into an integer, and comparing the vectors only when their hash values are equal. We design two heuristics to solve the oscillation problem.

—*Inverted Logic Insertion.* In this method, we add inverted logics to the outputs of some gates. In Figure 3(a), gate G_0 has output 0, and gates G_1, G_2, and G_3 require 0, 1, and 1 from gate G_0 to achieve smallest leakage values. Figure 3(b) illustrates how to add an OR gate between gate G_0 and its fanouts, so that all requirements are satisfied.[3] The OR gate in this example acts as an inverter when $Sleep = 1$, and does not change the functionality of the circuit when $Sleep = 0$. In our algorithm, adding an inverted logic to the output of a gate G is equivalent to splitting a tree circuit into two subtree circuits: the subtree rooted at gate G, and the rest of the original tree circuit. To avoid excessive area or delay increases due to the inverted logic insertion, we only add inverted logics to those gates with more than I_v fanouts (I_v is a user-controllable parameter). Inverted logic insertion perturbs the original circuit. Therefore it helps to remove the oscillation.

—*Dangling Assignment Perturbation.* If $\vec{D}(i) = \vec{D}(j)$ for some $i + 1 < j$, we perturb the assignment of $\vec{D}(j)$ so that our algorithm is able to jump out of the repeating loop. We compare $\vec{D}(j-1)$ and $\vec{D}(j)$ bit-by-bit, and change those bits of $\vec{D}(j)$ that are different with the corresponding bits of $\vec{D}(j-1)$. Assume the lth bits of $\vec{D}(j)$ and $\vec{D}(j-1)$ are different, let $n_1 = \sum_{k=i+1}^{j} \vec{D}(k)_l$, and $n_0 = j - i - n_1$.[4] The new value of the lth bit of $\vec{D}(j)$ is determined

[3]If the output of G is 1, we add an AND gate.

[4]Let $\vec{D}(j)_l$ denote the lth bits of $\vec{D}(j)$.

as follows: If $n_0 > n_1$, $\vec{D}(j)_l = 0$; if $n_0 < n_1$, $\vec{D}(j)_l = 1$; otherwise $\vec{D}(j)_l$ is assigned randomly.

Since the second method does not change the circuit, we apply it first. If the oscillation still exists, we apply the first heuristic. However, these techniques do not always resolve oscillation problems or guarantee converge. As we can see from Algorithm 1, our algorithm terminates after N_i iterations if these techniques fail.

3.6 Complexity of Overall Algorithm

The runtime of overall algorithm is summarized by the following theorem.

THEOREM 3.6. *The runtime of Algorithm* 1 *is* $O(nN_i^2)$*, where n is the number of gates in the circuit and* N_i *is the number of iterations in the algorithm.*

PROOF. Clearly, the algorithm consists of N_i iterations. In each iteration, the runtime of the dynamic programming algorithm is $O(n)$. The dangling input adjustment is a breadth-first search of the circuit, and its runtime is also $O(n)$. To check oscillations, we need to compare gate outputs in the current iteration with those in previous iterations, which takes at most $O(nN_i)$ time. As a result, the runtime of Algorithm 1 is $O(nN_i^2)$. □

According to experiments, the algorithm converges and terminates in 5 iterations on most circuits, which makes our algorithm much faster than stated in the previous theorem. In all our experiments, we set N_i to 20,[5] which means our algorithm is effectively a *linear* algorithm.

4. EXPERIMENTAL RESULTS

Our experiments are carried out on a SUN Ultra Sparc-10 server, and we use g++ to compile our programs. We follow the same synthesis flow [Sentovich et al. 1992] as in Yang and Qu [2005] and use the same MCNC91 combinational benchmark circuits [Yang 1991]. We use leakage data provided by Yuan and Qu [2005]. The purpose of using combinational circuits is for comparison. Our algorithm can also be used on sequential circuits, the results of which are similar to those on combinational circuits.

Our results are compared to those of two other algorithms [Rao et al. 2003; Yuan and Qu 2005]. In Rao et al. [2003], the authors introduced a fast heuristic method to find a good input vector (with no gate replacement considered). We skip our gate replacement step (*doRep* = 0 in Algorithm 2) to test the runtime of our input vector generation only. Our experiments indicate that our method is about 11 times faster than their approach, with slightly better leakage reduction. Yuan and Qu [2005] proposed a divide-and-conquer (DC, for short) algorithm with gate replacement, and showed better results as compared to the previous approaches. We will show that our algorithm is able to produce better leakage reduction compared to the DC algorithm, with several orders of magnitude speedup in runtime.

[5] $N_i = 20$ is enough to get a good result according to our experiments.

Table III. Comparison with Rao et al. [2003]

Ckt	$TmSv_o$	$Imprv_o$	$TmSv_p$	$Imprv_p$	Spdup
C1355	1269	6.47%	161.04	1.15%	7.88
C1908	533	−2.67%	63.15	2.11%	8.43
i10	730	4.54%	35.24	−0.44%	20.7
C2670	824	−0.05%	22.55	0.2%	36.56
C432	566	−3.57%	247.67	−1.49%	2.2
C499	1264	−1.88%	1098.22	0.14%	1.15
i5	919	7.83%	69.83	7.16%	13.16
C5315	809	−0.38%	24.48	−1.87%	33
C6288	1461	0.88%	458.81	2.97%	3.18
C7552	172	−0.94%	34.03	0.72%	5.06
C880	311	−0.68%	134.52	0.68%	2.31
my_add	533	1.48%	1318	−1.19%	0.40
average	783	0.92%	305.6	0.845%	11.18

In this table, $TmSv_o$ is the runtime savings of our algorithm against the random search; $TmSv_p$ is the runtime savings of Rao et al. [2003] against the random search; $Imprv_o$ is the leakage improvement of our algorithm against random search; $Imprv_p$ is the leakage improvement of Rao et al. [2003] against random search; Spdup is the runtime speedup of our algorithm against Rao et al. [2003].

Table III shows our comparison data with Rao et al. [2003]. In this experiment, we report the speedup of our algorithm against the random search method over 10000 input vectors for each circuit. For circuits with less than 13 inputs, we report our results against an exhaustive search method over the input vector space. All these parameters are the same as used by Rao et al. [2003], so that we have a fair comparison. As we can see from Table III, our algorithm runs 11 times faster on average than the heuristic method in Rao et al. [2003], on average.

To compare with the DC algorithm in Yuan and Qu [2005], we run our algorithm on 69 MCNC91 benchmark circuits provided by Yuan and Qu. For 26 small circuits with 22 or fewer primary inputs, we report our results against the exhaustive search. For 43 large circuits, we report our results against random search over 10000 input vectors. These parameters are the same as those of Yuan and Qu [2005]. The data of the DC algorithm was collected on the same type of machine we use (SUN Ultra Sparc-10 server). Table IV shows the average results for small, large, and all circuits. For 26 small circuits, our algorithm produces results 8% better than the DC algorithm; for 43 large circuits, our algorithm produces results 6% better than the DC algorithm with several orders of magnitude speedup; for all circuits, our algorithm outperforms the DC algorithm by 6.6%. In Table IV, the column $imprv_{best}$ shows the best leakage improvement over random search by setting $valveRep = 1$ in Algorithm 2. As shown in the table, we can achieve 10% more leakage improvement if we do not control the area and delay overhead. The average overheads of area and delay for $imprv_{best}$ are 18% and 4.4%, respectively. Table IV also indicates that our algorithm normally terminates in a few iterations (6 on average). Table V shows comparison results on the 10 largest combinational benchmark circuits. For large circuits, our algorithm outperforms the DC algorithm by 14.2% in

Table IV. Average Comparison with Yuan and Qu [2005]

	imprv_{best}	imprv_o	a_inc_o	time_o(s)	ite_o	imprv_p	a_inc_p	time_p(s)
small	34%	25%	7%	0.01	3.38	17%	9%	N/A
large	40%	30%	7%	0.10	7.19	24%	7%	510
average	38%	28%	7%	0.07	5.75	21.4%	7.8%	N/A

In this table, imprv_{best} is the best leakage improvement by our algorithm over random search (by setting $valveRep = 1$); imprv_o is the leakage improvement of our algorithm over random search with delay and area overhead control; a_inc_o is the area increase of our algorithm due to gate replacement; time_o is the runtime of our algorithm on average; the imprv_p, a_inc_p, and time_p fields show the corresponding results for the DC algorithm in Yuan and Qu [2005]; the field ite_o is the average number of iterations executed by our algorithm. The row *small* shows the average results for 26 small circuits; row *large* shows the average results for 43 large circuits; row *average* is the average results for all circuits.

Table V. Comparison with Yuan and Qu [2005] on Ten Largest Combinational Circuits

	imprv_{best}	imprv_o	a_inc_o	time_o(s)	imprv_p	a_inc_p	time_p(s)	Speedup
C6288	72%	30%	11%	0.82	8.8%	27.3%	398.7	486
C3540	43%	35.5%	5.8%	0.16	21.3%	2.1%	133.8	836
dalu	52%	42.1%	6.9%	0.11	23.2%	14.2%	194.9	1772
i8	48%	37%	10.8%	0.13	39.4%	6.3%	7591.3	58395
frg2	34%	22%	8.3%	0.07	28.4%	7.4%	176.5	2521
pair	39%	28.4%	4.2%	0.11	17.5%	12%	366	3327
C5315	56%	46%	6.7%	0.24	11.5%	15.1%	534.5	2227
C7552	46%	34.5%	6%	1.09	5.9%	16.1%	726	666
des	56%	44.7%	6.8%	0.40	45.7%	14.2%	8502.6	21257
i10	48%	38.5%	5.3%	0.25	14.3%	6.1%	162.8	651
average	49%	35.87%	7.18%	0.34	21.6%	12.8%	1878.71	9214

All fields have the same meanings as those of Table IV, except the field Speedup, which is the speedup of our algorithm over the DC algorithm.

terms of leakage reduction, and runs 214 to 24488 times faster. We have not shown the delays in both Table IV and Table V due to space limitations. For the 10 largest circuits, our algorithm with gate replacement increases critical-path delays by 1.38% on average, while the delay penalty reported in Yuan and Qu [2005] is limited by 5%. If performance is much more important than power consumption, we can disable gate replacement so that our algorithm does not affect critical-path delays at all.

5. CONCLUSIONS

The leakage power consumed by a circuit in sleep state can be reduced by applying a low-leakage vector. In this article, we proposed a fast algorithm that is able to find such a vector and apply the gate replacement technique simultaneously. Experiments revealed that our algorithm was able to produce leakage reduction results better than previous state-of-the-art approaches with several orders of magnitude speedup in runtime.

ACKNOWLEDGMENTS

We thank Lin Yuan and Gang Qu, the authors of Yuan and Qu [2005], for providing us the benchmark circuits, the leakage data, and their detailed experimental data.

REFERENCES

ABDOLLAHI, A., FALLAH, F., AND PEDRAM, M. 2004. Leakage current reduction in CMOS VLSI circuits by input vector control. *IEEE Trans. VLSI Syst. 12,* 2, 140–154.

ALOUL, F. A., HASSOUN, S., SAKALLAH, K. A., AND BLAAUW, D. 2002. Robust SAT-based search algorithm for leakage power reduction. In *Proceedings of the International Workshop on Power and Timing Modeling, Optimization and Simulation*, 167–177.

CHEN, Z. P., JOHNSONS, M., AND PEDRAM, M. 1998. Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 239–244.

CHOPRA, K. AND VRUDHULA, S. B. K. 2004. Implicit pseudo Boolean enumeration algorithms for input vector control. In *Proceedings the IEEE/ACM Design Automation Conference (DAC)*, 767–772.

CHOPRA, K. AND VRUDHULA, S. B. K. 2006. Efficient symbolic algorithms for computing the minimum and bounded leakage states. *IEEE Trans. Comput.-Aided Des. 25,* 12, 2820–2832.

GAO, F. AND HAYES, J. 2004. Exact and heuristic approaches to input vector control for leakage power reduction. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 527–532.

HALTER, J. AND NAJM, F. 1997. A gate-level leakage power reduction method for ultra-low-power CMOS circuits. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 475–478.

JOHNSON, M. C., SOMASEKHAR, D., AND ROY, K. 1999a. Leakage control with efficient use of transistor stacks in single threshold CMOS. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*, 442–445.

JOHNSON, M. C., SOMASEKHAR, D., AND ROY, K. 1999b. Models and algorithms for bounds on leakage in CMOS circuits. *IEEE Trans. Comput.-Aided Des. 18,* 6, 714–725.

KOBAYASHI, T. AND SAKURAI, T. 1994. Self-Adjusting threshold-voltage scheme (SATS) for low-voltage high-speed operation. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 271–274.

MUTOH, S., DOUSEKI, T., MATSU, Y., AOKI, T., SHIGEMATSU, S., AND YAMADA, J. 1995. 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS. *IEEE J. Solid-State Circuit. 30,* 8, 847–854.

NAFFZIGER, S., STACKHOUSE, B., AND GRUTKOWSKI, T. 2005. The implementation of a 2-core multi-threaded itanium-family processor. In *Proceedings of the International Solid State Circuits Conference*, 182–184.

RAO, R. M., BURNS, J. L., AND BRWN, R. B. 2003. A heuristic to determine low leakage sleep state vectors for CMOS combinational circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 689–692.

SENTOVICH, E. M., SINGH, K. J., LAVAGNO, L. MOON, C., MURGAI, R., SALDANHA, A., SAVOJ, H., STEPHAN, P. R., AND BRAYTON, R. K. 1992. SIS: A system for sequential circuit synthesis. Electronics Research Laboratory Memo UCB/ERL M92/41. University of California at Berkeley. California.

WEI, L. Q., CHEN, Z., JOHANSON, M., ROY, K., AND DE, V. 1998. Design and optimization of low voltage high performance dual threshold CMOS circuits. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*, 489–494.

YANG, S. 1991. Logic synthesis and optimization benchmarks user guide, version 3.0. ftp://mcnc.mcnc.org.

YE, Y., BORKAR, S., AND DE, V. 1998. A new technique for standby leakage reduction in high-performance circuits. In *Proceedings of the IEEE Symposium on VLSI Circuits*, 40–41.

YUAN, L. AND QU, G. 2005. Enhanced leakage reduction technique by gate replacement. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*, 47–50.

Received September 2007; accepted September 2007