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

Lei Cheng, Liang Deng, Deming Chen, and Martin D.F. Wong
Coordinated Science Laboratory, UIUC
1308 West Main St. Urbana, IL 61801
University of Illinois at Urbana-Champaign
{lcheng1,ldeng,dchen,mdfwong}@uiuc.edu

ABSTRACT

Input vector control (IVC) technique is based on the observation that the leakage current in a CMOS logic gate depends on the gate input state, and a good input vector is able to minimize the leakage when the circuit is in the sleep mode. The gate replacement technique is a very effective method to further reduce the leakage current. In this paper, we propose a fast 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-Aided Engineering]: Computer Aided Design

General Terms: Algorithm, Performance

Keywords: Input vector control, leakage reduction, gate replacement

1. INTRODUCTION

Leakage reduction techniques have been proposed previously on device, circuit and system levels. Dual threshold voltage (V_t) transistors [2] and body biasing [3] techniques change threshold voltage, and reduce leakage effectively. Later on, sleeping transistor [5] and gated-V_{dd} [6] techniques were introduced to reduce the leakage power. Input vector control (IVC) is another way to efficiently reduce leakage power [4, 5, 7, 8]. For large circuits with deep levels, IVC becomes less effective because the controllability of internal nodes is lower. Gate replacement technique can be used to improve the controllability.

Most of the proposed heuristics for IVC become slow for large circuits. Considering gate replacement, the computation complexity increases further. Existing methods for gate replacement is formulated assuming the input vector is known, and gate replacement is carried out after input vector generation. However, these two techniques actively interact with each other. Without consideration of the interaction, the leakage reduction results are obviously not optimal.

In this paper, we present an $O(n)$ algorithm to solve the IVC and gate replacement problems simultaneously. A dynamic programming based-algorithm is used for making fast evaluation on input vectors, as well as 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 algorithm.
- We propose a linear time dynamic programming algorithm producing the optimum input vector with simultaneous gate replacement for tree circuits.
- We propose an iterative method that can produce the low leakage input vector and gate replacement solution for a circuit very fast.
- Our algorithm produces better results than the previous state-of-the-art IVC algorithm with gate replacement [1] with several orders of magnitude speedup in runtime.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

2. PRELIMINARIES AND RELATED WORKS

2.1 Input Vector Control

Input vector control (IVC) methods were first proposed in [7, 4]. This technique is based on the observation that the leakage current in a CMOS logic gate is dependent on the gate input state. However, finding such an optimal input vector is NP-hard [9]. Both SAT-based algorithm [10] and integer linear programming based algorithm [11] were proposed previously. A random search algorithm was proposed by Halter and Najm [7]. It was reported that a random search over 10000 vectors gave us over 99% confidence that less than 0.5% of the vector population would have a leakage lower than the minimum leakage value observed from the random search [7, 8]. In [1], a genetic algorithm was proposed to find the input vector with gate replacement, which requires long runtime for large circuits due to the nature of the genetic algorithm.

2.2 Gate Replacement

When paths in circuits become deeper, the IVC techniques become less effective because gates with deep levels are harder to be affected by the input vector. One way to handle this problem is the *gate replacement* method used in [1]. In [1], an optimum input vector was first produced by a dynamic programming algorithm, followed by a heuristic gate replacement algorithm. 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, which changes the leakage current of gates it fans out to. In this paper, these two methods are considered simultaneously through a dynamic programming-based technique.

The gate replacement technique is to replace a gate $G(\vec{x})$ by another gate $\tilde{G}(\vec{x}, Sleep)$ where $\vec{x}$ is the input vector at G, such that: 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 the standby mode ($Sleep = 1$).

Figure 2 shows how to replace an NAND2 gate. According to the leakage data, the leakage of NAND2 with the

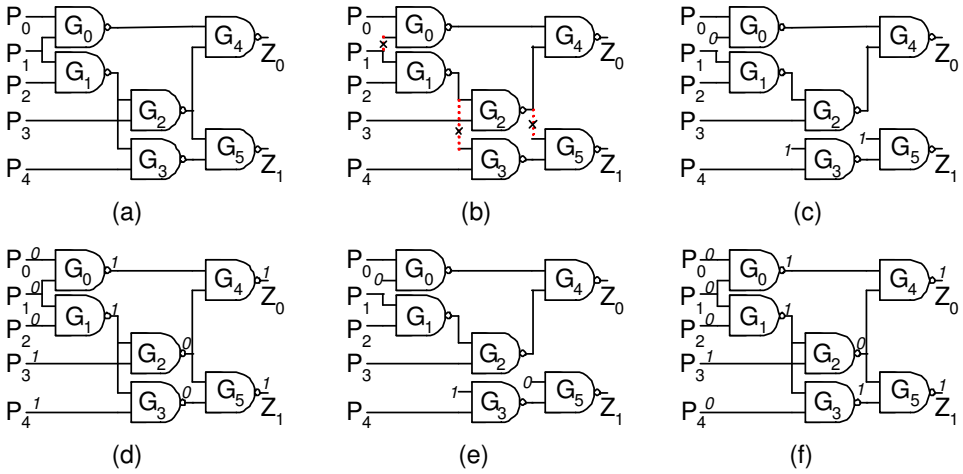

Figure 1: Step by step illustration of our algorithm on the circuit $C17$.

vector 11 is $454.50nA$, and it is reduced to $94.87nA$ after the NAND2 gate is replaced by an NAND3 gate (see [1]).

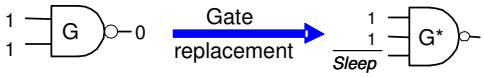

Figure 2: Gate replacement example.

3. SIMULTANEOUS INPUT VECTOR GENERATION WITH GATE REPLACEMENT

In this section, we present our algorithm of 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 non primary 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 in Fig. 1.(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. The details will be presented in the following subsections.

3.1 An Example

Fig. 1 provides an example of running our algorithm on a simple MCNC91 benchmark circuit $C17$ (Fig. 1.(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, G_2 and G_5 (Fig. 1.(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 (Fig. 1.(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 (Fig. 1.(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 next iteration. In Fig. 1.(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 (Fig. 1.(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 $C17$.

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

Algorithm 1: Overall algorithm

Input: $\{G_1, \ldots, G_n\}$: a circuit with n gates;
$Iterations$: 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 Iterations do
    Perform dynamic programming algorithm for each tree;
    Adjust the dangling assignment;
    CheckOscillations;
    if solutions converge then
        break;
end
```

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. Let's $LP(i)$ denote the longest path from gate G_i to one primary output. We try to keep connections for those gates with large LP values, because the larger the $LP(i)$, the larger the number of gates affected by gate G_i. If LP values are the same, we prefer to keep connections for gates with large number of fanouts.

Note that our link-deletion based transformation is different from the tree decomposition method in [1], in which every gate with multiple fanouts is the root of a decomposed tree. Their method produces a large number of trees,

Table 1: 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 $\vec{x}$

and the size of each tree is very small. Large number of small trees makes the dynamic programming algorithm on trees very ineffective, because the interactions among trees are more important in this case. In our algorithm, only the primary outputs of the circuit are the roots of trees.

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. To estimate the gate outputs at the beginning of our algorithm, 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 $G_1 \dots G_k$, 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.

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 the minimum leakage. First, we generate a list of gates in the topological order. Then we evaluate gates 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 is the same as that used in [1].

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 1 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 the minimum leakage for subtree rooted at the gate G_i is generated by combining the input vectors of G_i's fanin trees. In the algorithm, the variable *valveRep* is used to control the area and delay of the final circuit. Replacing gates increases the area and delay because a 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 $valveRep = 1$, our algorithm produces the minimum leakage vector for a tree circuit with gate replacement. The calculation of $\vec{R}$ at the end of the algorithm is trial, and we omit its details in this paper.

Algorithm 2: Algorithm producing optimum input vector with gate replacement for a tree circuit

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

```
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 Rep 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 1. *Algorithm 2 produces the minimum leakage vector for a tree circuit with simultaneous gate replacement when* $valveRep = 1$.

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, so we have the following theorem.

THEOREM 2. *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.*

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. Whenever we detect that a dangling assignment is equal to a previous one, we perturb it according to some heuristic. Basically, this heuristic assigns a value 0 or 1 to a primary input based on whether 0 or 1 appears most often for this input between the two repeated iterations. For example, let us assume iteration 2 and iteration 8 are repeated, and the value 0 has been assigned to the input I four times, and the value 1 has been assigned to I twice from iteration 2 to iteration 7, then we will assign value 0 to the input I in the iteration 8 to break the oscillation.

Table 2: Average comparison with paper [1]. 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 and d_inc_o are the area and delay increases of our algorithm due to gate replacement; time_o is the runtime of our algorithm on average; imprv_p, a_inc_p, d_inc_p and time_p fields show the corresponding results for the DC algorithm in [1]; 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; the row *large* shows the average results for 43 large circuits; the row *average* is the average results for all circuits.

	imprv_{best}	imprv_o	a_inc_o	d_inc_o	time_o(s)	ite_o	imprv_p	a_inc_p	d_inc_p	time_p(s)
small	34%	25%	7%	2.4%	0.01	3.38	17%	9%	<5%	N/A
large	40%	30%	7%	1.8%	0.10	7.19	24%	7%	<5%	510
average	38%	28%	7%	2%	0.07	5.75	21.4%	7.8%	<5%	N/A

Table 3: Comparison with paper [1] on ten largest combinational circuits. All the columns have the same meanings as those of table 2, except the column Speedup, which is the speedup of algorithm over the DC algorithm.

	imprv_{best}	imprv_o	a_inc_o	d_inc_o	time_o(s)	imprv_p	a_inc_p	d_inc_p	time_p(s)	Speedup
C6288	72%	30%	11%	2.6%	0.82	8.8%	27.3%	<5%	398.7	486
C3540	43%	35.5%	5.8%	1.5%	0.16	21.3%	2.1%	<5%	133.8	836
dalu	52%	42.1%	6.9%	1%	0.11	23.2%	14.2%	<5%	194.9	1772
i8	48%	37%	10.8%	2.8%	0.13	39.4%	6.3%	<5%	7591.3	58395
frg2	34%	22%	8.3%	0.8%	0.07	28.4%	7.4%	<5%	176.5	2521
pair	39%	28.4%	4.2%	1.1%	0.11	17.5%	12%	<5%	366	3327
C5315	56%	46%	6.7%	1.6%	0.24	11.5%	15.1%	<5%	534.5	2227
C7552	46%	34.5%	6%	0.7%	1.09	5.9%	16.1%	<5%	726	666
des	56%	44.7%	6.8%	0.8%	0.40	45.7%	14.2%	<5%	8502.6	21257
i10	48%	38.5%	5.3%	0.9%	0.25	14.3%	6.1%	<5%	162.8	651
average	49%	35.87%	7.18%	1.38%	0.34	21.6%	12.8%	<5%	1878.71	9214

4. EXPERIMENTAL RESULTS

In our experiments, we follow the same synthesis flow using SIS [14] as in [1] and use the combinational benchmark circuits provided by the authors of [1]. The purpose of using combinational circuits is for comparison. Our algorithm can also be used on sequential circuits, and the results on sequential circuits are similar to those on combinational circuits.

To compare with the divide and conquer (DC) algorithm in [1], we run our algorithm on 69 MCNC91 benchmark circuits provided by the authors of [1]. 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 with [1]. The data of the DC algorithm were collected on the same type of machine we use (SUN Ultra Sparc-10 server). Table 2 shows the average results for small circuits, large circuits, and all of them. For 26 small circuits, our algorithm produces results 8% better than the DC algorithm; for 43 large circuits, our algorithm produce results 6% better than the DC algorithm with several orders of magnitude of speedup; for all circuits, our algorithm outperforms the DC algorithm by 6.6%. The field 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 the area and delay for imprv_{best} are 18% and 4.4% respectively. Table 2 also indicates that our algorithm normally terminates in a few iterations (6 on average). Table 3 shows comparison results on ten largest combinational benchmark circuits. For large circuits, our algorithm outperforms the DC algorithm by 14.2% in term of leakage reduction, and runs 214 to 24488 times faster.

5. CONCLUSIONS AND ACKNOWLEDGEMENTS

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

We thank Lin Yuan and Gang Qu, the authors of [1], for providing us the benchmark circuits, the leakage data, and their detailed experimental data.

6. REFERENCES

[1] L. Yuan and G. Qu. Enhanced leakage reduction technique by gate replacement. In *DAC*, pages 47–50, 2005.

[2] L. Q. Wei et al. Design and optimization of low voltage high performance dual threshold CMOS circuits. In *DAC*, pages 489–494, 1998.

[3] T. Kobayashi and T. Sakurai. Self-adjusting threshold-voltage scheme (SATS) for low-voltage high-speed operation. In *CICC*, pages 271–274, 1994.

[4] Y. Ye, S. Borkar, and V. De. A new technique for standby leakage reduction in high-performance circuits. In *Symp. VLSI Circuits*, pages 40–41, 1998.

[5] M. C. Johnson, D. Somasekhar, and K. Roy. Leakage control with efficient use of transistor stacks in single threshold CMOS. In *DAC*, pages 442–445, 1999.

[6] S. Mutoh et al. 1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS. *JSSC*, 30(8):847–854, 1995.

[7] J. Halter and F. Najm. A gate-level leakage power reduction method for ultra-low-power CMOS circuits. In *CICC*, pages 475–478, 1997.

[8] R. M. Rao et al. A heuristic to determine low leakage sleep state vectors for CMOS combinational circuits. In *ICCAD*, pages 689–692, 2003.

[9] M. C. Johnson, D. Somasekhar, and K. Roy. Models and algorithms for bounds on leakage in CMOS circuits. *IEEE Trans. on CAD*, 18(6):714–725, 1999.

[10] F. A. Aloul et al. Robust SAT-based search algorithm for leakage power reduction. In *PATMOS*, pages 167–177, 2002.

[11] F. Gao and J.P. Hayes. Exact and heuristic approaches to input vector control for leakage power reduction. In *ICCAD*, pages 527 – 532, 2004.

[12] A. Abdollahi et al. Leakage current reduction in CMOS VLSI circuits by input vector control. *IEEE Trans. on VLSI Systems*, 12(2):140–154, 2004.

[13] Z. P. Chen et al. Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks. In *ISLPED*, pages 239–244, 1998.

[14] E. Sentovich et al. SIS: A system for sequential circuit synthesis. In *Electronics Research Laboratory Memorandum, U.C.Berkeley*. No. UCB/ERL M92/41.