Enhanced Leakage Reduction Technique by Gate Replacement

Lin Yuan and Gang Qu
Electrical and Computer Engineering Department and Institute for Advanced Computer Studies
University of Maryland, College Park, MD, 20742 USA
{yuanl, gangqu}@eng.umd.edu

Abstract

Input vector control (IVC) technique utilizes the stack effect in CMOS circuit to apply the minimum leakage vector (MLV) to the circuit at the sleep mode to reduce leakage. Additional logic gates can be inserted as control points to make it more effective. In this paper, we propose a gate replacement technique that further enhances the leakage reduction. The basic idea is to replace a gate that is in its worst leakage state by another library gate while keeping the circuit's correct functionality at the active mode. We also develop a divide-and-conquer approach that integrates a fast gate replacement heuristic, an optimal MLV search strategy for tree circuit, and a genetic algorithm to connect the tree circuits. We conduct experiments on the MCNC91 benchmark circuits. The results reveal that our technique can reduce additional 10% to 24% leakage over the best known IVC methods and the optimal MLV with no delay penalty and little area increase.

Categories and Subject Descriptors: B.6.3 Design Aids

General Terms: Algorithms, Performance, Design.

Keywords: Leakage reduction, gate replacement, MLV.

1. INTRODUCTION

Due to the continued scaling of technology and supply/threshold voltage, leakage power has become more and more significant in the power dissipation of today's CMOS circuits. For example, it is projected that subthreshold leakage power can contribute as much as 42% of the total power in the 90nm process generation [8]. Many leakage reduction techniques have been proposed recently. Among them, the input vector control (IVC) technique is very popular because it is effective to reduce leakage at runtime and does not require additional technology [4].

The IVC technique is based on the transistor stack effect: a CMOS gate's leakage current varies dramatically with the input vector applied to the gate [7]. A significant leakage current reduction in a circuit can be achieved if an input vector that causes minimum total leakage (MLV) can be found at the primary input of the circuit. This problem, however, is NP-hard [7]. A random search is proposed to find the MLV among thousands of randomly generated input vectors [6]. Aloul et al. [2] and Gao et al. [5] formulated the problem using SAT and ILP respectively.

We say that a logic gate is at its worst leakage state (WLS) if its input yields the highest leakage current. Regardless of the primary input vector, a large number of gates are at WLS, particularly when the circuit has high logic depth. For each of the 69 MCNC91 benchmark circuits, when we apply the optimal (or sub-optimal) MLVs to these circuits, 16% of the gates on average remain at WLS, producing more than 40% of the circuit's total leakage. A detailed report can be found in Section 4.

Therefore, instead of fine-tuning the existing algorithms to find MLV, we formulate and study the **MLV+ problem** that seeks to *modify a given circuit and determine an input vector such that the correct functionality is maintained when the circuit is active and the leakage is minimized when the circuit is in standby mode*.

The motivation of modifying the circuit is to reduce the number of the gates at WLS in order to achieve low leakage. It is based on the following observation: take the NAND2 gate for example, the worst leakage current (454.50 nA for a 0.18μm technology) happens when both its inputs are logic 1 at the standby mode. However, if we replace it by a NAND3 gate and let the additional input signal to be 1 at active mode and 0 at standby mode (this can be trivially achieved by the complement of the $SLEEP$ signal), then the input to this NAND3 at standby mode will be $\{110\}$ which yields 94.87nA leakage current, a reduction of about 80%!

Our technique is conceptually different from the internal pin control technique proposed by Abdollahi et al. [1]. First, their purpose of modifying a gate G is to produce the low-leakage input for gates at G's fanout; we aim to reduce leakage current at G itself. Second, in their technique, the gate structure is changed and therefore new library gates are needed; however, our algorithm is restricted to replace gates within the existing technology library. Finally, they treat each internal pin as potential places to control, which results in large complexity; we first partition the circuit into trees and only control values at the root of each tree.

2. GATE REPLACEMENT TECHNIQUE

Our proposed gate replacement technique replaces a gate $G(\vec{x})$ at WLS by another gate $\tilde{G}(\vec{x}, SLEEP)$ in the library, 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 $G(\vec{x})$ at WLS when the circuit is in standby mode ($SLEEP = 1$).

The first condition guarantees the correct functionality of the circuit at active mode and the second condition reduces the leakage on gate G at the standby mode. Note that the replacement of gate

G by $\tilde{G}$ may change the output of this gate. Although we do not need to maintain the circuit's functionality at the standby mode, this change may affect the leakage of other gates and should be carefully considered. See Figure 1 for the example when a NAND2 gate G is replaced by NAND3.

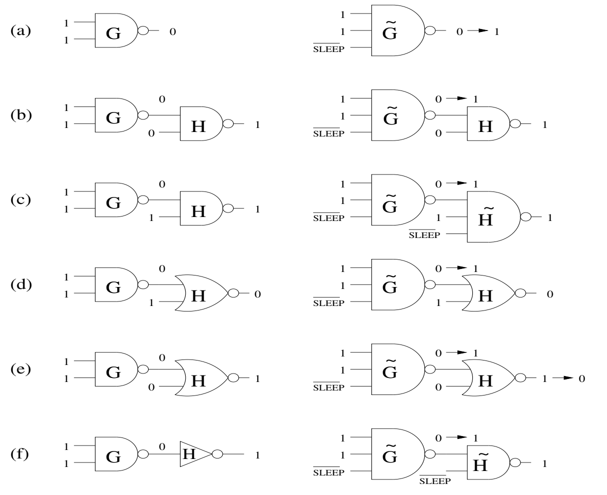

Figure 1: Gate replacement and its fanout change.

A Fast Gate Replacement Algorithm Figure 2 gives the pseudocode of our proposed fast gate replacement algorithm. For a circuit with a given input vector, we visit the gates in the circuit by the topological order. We skip all the gates that are not at WLS and the visited (or marked) gates (line 16) until we find a new gate G_i at WLS (line 2). Lines 3-9 find a subset of gates $\mathcal{S}$ and temporarily replace them. $\mathcal{S}$ includes all the unmarked gates whose leakage and/or output is affected by the replacement we attempt to do on gate G_i. We then compute the total leakage change caused by the replacement of gates in $\mathcal{S}$ (line 10) and adopt these replacements if there is a leakage reduction (lines 11-13). Otherwise, we simply mark gate G_i as visited and do not make any replacement (line 14). We then look for the next unmarked gate at WLS and this procedure stops when all the gates in the circuits are marked.

```
Input: ${G_1, G_2, \cdots}$: gates in a circuit sorted topologically,
       ${x_1, x_2, \cdots}$: an input vector, $SLEEP$: the sleep signal.
Output: a circuit of the same functionality when $SLEEP = 0$ and
        with less leakage when $SLEEP = 1$.
Gate Replacement Algorithm:
 1. for each gate $G_i \in \{G_1, G_2, \cdots\}$
 2.   if ($G_i$ is at WLS and not marked)
 3.     include $G_i$ in the selection $\mathcal{S}$;
 4.     while (there is new addition to $\mathcal{S}$)
 5.       for each newly selected gate $G$ in $\mathcal{S}$
 6.         if (there exists library gate $\tilde{G}$ meets conditions 1 and 2)
 7.           temporarily replace $G$ by $\tilde{G}$;
 8.           if (output of $G$ is changed due to this replacement)
 9.             include $G$'s unmarked fanout gate $G_j$ in $\mathcal{S}$;
10.     compute the total leakage change of gates in $\mathcal{S}$;
11.     if (there is leakage reduction)
12.       mark all gates $G_j$ in the selection $\mathcal{S}$;
13.       make the replacements in lines 7,9,or 10 permanent;
14.     else mark gate $G_i$ only;
15.     empty the selection $\mathcal{S}$;
16.   else mark $G_i$ if it has not been marked yet;
```

Figure 2: Pseudocode of the gate replacement algorithm.

Correctness: The topological order guarantees that when we find a gate at its WLS, all its predecessors have already been considered. The replacement at line 7 ensures that the functionality will not change at the active mode. The subset $\mathcal{S}$ constructed in the **while** loop (lines 3-9) is the *transitive closure* of gates that are affected by the replacement action at gate G_i. Therefore, we only need to compute the leakage change on gates within $\mathcal{S}$. We make the replacement only when this leakage change is in favor of us, so the new circuit will have less leakage in standby mode.

Complexity: Let n be the number of gates in the circuit. The **for** loop is linear to n. Inside the for loop, the computation of leakage change (line 10) and the marking of all gates in $\mathcal{S}$ is linear to $|\mathcal{S}|$, the number of gates in $\mathcal{S}$. The **while** loop (lines 3-9) stops when there is no new addition to $\mathcal{S}$ and this will be executed no more than $|\mathcal{S}|$ times. As we have discussed in section 3.1 (see Figure 1), in most cases, $\mathcal{S}$ includes only G and its fanout gates. However, it may include all the gates of the circuit in cases similar to Figure 1 (e) and so $|\mathcal{S}|$ cannot be bounded by any constant. That is, $|\mathcal{S}|$ is $O(n)$ in the worst case and $O(k)$ on average, where k is the maximal fanout of the gates in the circuit. Consequently, the complexity of this gate replacement algorithm is $O(n^2)$ in the worst case and $O(kn)$ on average.

3. DIVIDE AND CONQUER ALGORITHM

In this section, we describe a divide-and-conquer approach to solve the MLV+ problem. First, we decompose a given circuit into tree circuits where the MLV can be determined optimally by a dynamic programming approach. Then we connect these tree circuits by a genetic algorithm that utilizes the gate replacement algorithm.

3.1 Optimal MLV for Tree Circuits

A tree circuit is a single output circuit in which each gate, except the output, feeds exactly one other gate. Any general combinational circuit can be trivially decomposed into non-overlapping tree circuits. Moreover, the topological order of the gates in the original circuit will be kept in the tree circuits. Therefore, we consider a tree circuit with gates $\{G_1, G_2, \cdots, G_n\}$ in the topological order.

Let $L(G_i(\vec{x}))$ be the leakage current in gate G_i when vector $\vec{x}$ is applied as G_i's fanins. Let $\vec{V}(i, z)$ be the input vector to the subtree rooted at gate G_i such that 1) the output of G_i is z and 2) the leakage in the subtree rooted at G_i, denoted by $LK(i, z)$, is minimized. We develop a dynamic programming approach to compute the pairs $(LK(i, 0), \vec{V}(i, 0))$ and $(LK(i, 1), \vec{V}(i, 1))$ for each gate G_i. The MLV for the tree circuit with gates $\{G_1, G_2, \cdots, G_n\}$ can then be determined conveniently.

We use the notation G_0 for any input signal of the tree and start the dynamic programming by defining

$$LK(0, z) = 0, \qquad \vec{V}(0, z) = z \tag{1}$$

For any logic gate G_i, let $\{x_{i_1}, x_{i_2}, \cdots, x_{i_t}\}$ be its fanins from gates $\{G_{i_1}, G_{i_2}, \cdots, G_{i_t}\}$ respectively. To compute $LK(i, z)$, we need to consider all the possible combination of fanins $\{x_{i_1}, \cdots, x_{i_t}\}$ that produces output z at gate G_i. For each such case, the minimum leakage in the subtree rooted at G_i is the sum of leakage at gate G_i and the minimum leakage at each of its fanin gate G_{i_j} with output x_{i_j}, $LK(i_j, x_{i_j})$. Thus, we have

$$LK(i, z) = \min_{\forall \vec{x}, s.t. G_i \text{ outputs } z} \left(L(G_i(\vec{x})) + \sum_{j=1}^{t} LK(i_j, x_{i_j})\right) \tag{2}$$

Note that G_i's fanin gates $\{G_{i_1}, \cdots, G_{i_t}\}$ do not share any common input signals due to the structure of tree circuit. Assuming that $LK(i, z)$ is achieved with fanins $\{x^0_{i_1}, \cdots, x^0_{i_t}\}$, we have

$$\vec{V}(i, z) = \cup_{j=1}^{t} \vec{V}(i_j, x^0_{i_j}) \tag{3}$$

The dynamic programming procedure stops at the root gate G_n after we compute $(LK(n,z), \vec{V}(n,z))$ for $z = 0$ and $z = 1$. The minimum leakage of the tree circuit with gates $\{G_1, \cdots, G_n\}$ is

$$\min\{LK(n,0), LK(n,1)\} \tag{4}$$

and the MLV will be either $\vec{V}(n,0)$ or $\vec{V}(n,1)$ accordingly. Finally, we perform the gate replacement algorithm to further reduce leakage under the obtained MLV.

Complexity: Equations (1) and (4) take constant time. We need to compute $(LK(i,0), \vec{V}(i,0))$ and $(LK(i,1), \vec{V}(i,1))$ for each gate G_i in equations (2) and (3). This requires the enumeration of all the 2^t different combinations of G_i's t inputs. For the first time, we need to perform t additions in equation (2). If we enumerate the rest $2^t - 1$ cases following a Gray code, we only need to update $L(G_i(\vec{x}))$, replace one $LK(i_j, x_{i_j})$ and compare the result with the current minimum leakage, a total of five operations. Therefore, we need $t + 5 \cdot (2^t - 1)$ operations for each G_i and this gives a complexity of $O(K \cdot n)$, where K is a constant depending on the largest number of fanins in the circuit.

3.2 Connecting the Tree Circuits

In the previous phase, we have determined the output and required input for each individual tree circuit to yield the minimum leakage. The goal of this phase is to combine all the tree circuits to solve the MLV+ problem for the original circuit. The root of each tree circuit has multiple fanouts that go to other tree circuits as input. Since we treat the tree circuits independently, *conflict* occurs if the output of a tree circuit and the value required by its fanout gates are not consistent.

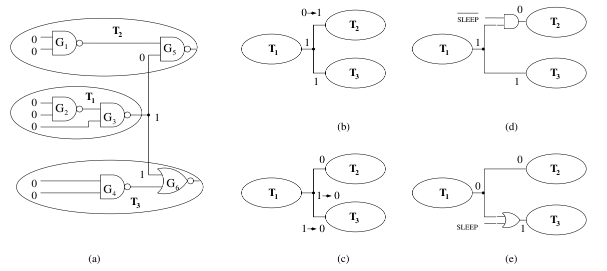

Figure 3: Resolving the conflict in connecting tree circuits.

For example, in Figure 3 (a), the circuit is decomposed into three tree circuits T_1, T_2 and T_3. T_1 outputs '1' when its MLV is applied, while T_2 and T_3 require '0' and '1' from T_1 in their respective MLVs. So we have a conflict. There are basically three ways to resolve this conflict:

(I) enforcing T_1's output at all the fanout gates (Figure 3 (b));

(II) changing T_1's output and enforcing this new value at all the fanout gates (Figure 3 (c));

(III) inserting an AND gate to allow them to be inconsistent (Figure 3 (d)). Similarly, if T_1 output '0' and some of its fanouts require '1', we can add an OR gate as shown in Figure 3 (e)).

For each solution, we need to re-evaluate the circuit's total leakage. In (I), this requires the re-computing of the minimum leakage and the MLV for tree circuit T_2 under the condition that its input from T_1 is constant '1'. The dynamic programming algorithm in Section 4.1 can be trivially modified for this purpose. In (II), we need to do the same procedure for tree circuit T_3. Besides, we have to replace the pair $\{LK(n,1), \vec{V}(n,1)\}$ for tree circuit T_1 by $\{LK(n,0), \vec{V}(n,0)\}$.

Both (I) and (II) resolve the conflict by sacrificing the minimum leakage of tree circuits under the provably optimal MLV. In (III), we successfully connect the tree circuits while preserving the minimum leakage and MLV for each tree with the help of the $SLEEP$ signal-controled AND or OR gates. The cost is that we have to add the leakage of the inserted AND or OR gate into the total leakage. We mention that this gate addition also preserves the correctness of the circuit at active mode when $SLEEP$=0.

It is simple to make a decision on which method to adopt to resolve a single conflict: use the one that gives the minimum leakage. However, the decision at one conflict may affect the existence of conflict at other places in the circuit. For example, method (I) in Figure 3 (b) could change the output of tree T_2 and directly affect whether there is a conflict at the root of T_2.

We use a genetic algorithm (GA) to resolve the conflicts and connect all the tree circuits. A solution by the GA is in the form of a binary bit stream, each bit indicates whether there is a conflict at the root of a tree and which method to use to resolve it. In particular, a '1' means there is a conflict and method (III) should be used; a '0' means that there is either no conflict or we should use the better one of methods (I) and (II) to resolve the conflict.

The GA follows a standard routine where we start with a population of N random bit streams (referred to as *chromosomes*). Based on each bit stream, we resolve the conflict, apply the dynamic programming algorithm in Section 3.1 to re-compute the minimum leakage of a tree circuit when methods (I) and (II) are used, run the gate replacement algorithm in Figure 2 on the entire circuit, and compute the circuit's total leakage. The *fitness* for a bit stream is calculated from the leakage value. The smaller the leakage, the larger the *fitness*. We sort all the chromosomes according to their fitness and create the next generation by the *roulette wheel* method. In this method, the probability that a *chromosome* is selected as one of the two parents is proportional to its fitness. *Crossover*, which refers to the exchange of substrings in two chromosomes, is performed among parents to produce children. A simple *mutation* operation, which flips a bit in the chromosome at the *bit mutation rate*, is also used. The GA continues to generate a total of N new chromosomes and starts for the next generation. This process repeats for certain number of times (50 in our simulation) and the best chromosome is returned as the optimal solution.

4. EXPERIMENTAL RESULTS

We implemented the gate replacement and divide-and-conquer techniques in SIS environment and applied them on 69 MCNC91 benchmark circuits. Each circuit is synthesized and mapped to a 0.18 μm technology library. We use Cadence Spectre to simulate the leakage current for all the library gates under every possible input vector. The supply voltage and threshold voltage are 1.5V and 0.2V, respectively.

The goal of our simulation is to demonstrate the effectiveness of the proposed techniques in leakage reduction. Our results are compared with traditional input vector control methods in terms of leakage saving, run time, area and delay penalty. We conducted experiments on 69 benchmarks including 26 small circuits with 22 or fewer primary inputs (Table 1) and 43 large circuits (Figure 4). For each small circuit, we find its optimal MLV by exhaustive search. For large circuits, we choose the best MLV from 10,000 distinct random input vectors. It is reported that this will ensure us with a 99% confidence that the vectors with less leakage is less than 0.5% of the entire vector population [6, 9]. To have a fair comparison with [1], we also collect the average leakage of 1,000 random input vectors for each large circuit.

Table 1 reports the results for the 26 small circuits. Column 4

Table 1: Results on 26 small circuits with 22 or less PIs.

circuit	pi	gate	exhaustive		gate replace			divide-and-conquer			
	#	#	leak (nA)	wls	imprv	wls	ar inc	imprv	# tree	# cg	ar inc
b1	3	13	2195.1	23%	2%	15%	5%	2%	5	0	5%
cm42a	4	25	2941.3	0%	0%	0%	0%	8%	18	1	8%
C17	5	6	831.1	17%	43%	0%	17%	43%	4	0	17%
cm82a	5	28	5017.2	21%	29%	4%	12%	40%	10	1	18%
decod	5	22	1920.7	0%	0%	0%	0%	8%	21	1	3%
cm138a	6	19	1760.4	0%	0%	0%	0%	1%	12	1	5%
z4ml	7	66	12245.6	24%	25%	11%	11%	37%	20	3	17%
f51m	8	136	26037.8	26%	37%	7%	12%	48%	25	4	14%
9symml	9	166	34017.5	26%	20%	17%	5%	38%	18	13	14%
alu2	10	356	64153.0	21%	2%	20%	0%	21%	89	24	11%
x2	10	44	6158.8	9%	15%	2%	3%	12%	18	4	10%
cm85a	11	38	4925.2	8%	14%	3%	3%	13%	16	0	3%
cm151a	12	34	5745.0	24%	9%	18%	4%	3%	5	1	5%
alu4	14	728	133126.5	25%	1%	21%	1%	15%	166	49	10%
cm162a	14	45	6946.6	18%	2%	9%	3%	0%	13	2	12%
cu	14	49	6181.7	12%	16%	6%	2%	9%	21	3	7%
cm163a	16	43	6375.9	19%	2%	9%	3%	1%	11	2	13%
cmb	16	42	5404.7	10%	11%	5%	2%	4%	8	1	6%
parity	16	75	12763.7	20%	11%	15%	5%	15%	15	5	20%
pm1	16	39	3473.9	3%	0%	0%	1%	-2%	16	1	3%
t481	16	1945	251183.8	2%	1%	1%	0%	26%	17	3	1%
tcon	17	41	6491.0	20%	43%	0%	14%	41%	9	1	17%
pcle	19	74	12594.1	20%	32%	4%	6%	32%	22	0	6%
sct	19	92	11811.3	18%	14%	9%	4%	10%	24	4	6%
cc	21	48	5822.5	13%	6%	10%	1%	6%	22	0	1%
cm150a	21	72	12269.5	15%	4%	14%	1%	1%	9	5	10%
Average				15%	13%	8%	4%	17%			9%

lists the leakage current for each circuit when the best MLV is applied. Even in this case, an average of 15% of the gates are at WLS as shown in column 5. The fast gate replacement algorithm is able to remove about half of these gates from their WLS (column 7). This results in a 13% leakage reduction with only 4% area increase (columns 6 and 8). We mention that we restrict ourselves to replace only gates that are off-critical path by gates from the library. This leaves 8% of the gates in the circuits at their WLS, but it also guarantees us there is no delay overhead.

The last four columns show that the divide-and-conquer algorithm gives a 17% leakage reduction over the best MLV at the cost of 9% more area. We incorporate delay constraints in the genetic algorithm to ensure that the delay overhead to be within 5%. The two columns in the middle are the number of tree circuits in each case and the number of control gates we have used to connect these trees. Only in three cases, we have inserted more than five control gates. Note that the addition of control gates may decrease the delay because it reduces the fanouts of the gate. The area increase comes from the addition of control gates and the replacement of "smaller" gates by "bigger" library gates.

Figure 4 reports the results on the 43 circuits (x-axis) with 22 PIs or more. We replace the infeasible exhaustive search by the best solution from a random search of 10K different input vectors. We restrict the fast gate replacement algorithm to gates off critical path only to avoid delay overhead. For the divide-and-conquer approach, we set the maximal delay increase to be 5%.

The three curves at the bottom of Figure 4 give the ratio of WLS gates. On average, the 10K random search has **17%** gates at WLS (wls_orig); the proposed fast gate replacement and divide-and-conquer techniques reduce this ratio to **11%** (wls_gr) and **9%** (wls_dc), respectively. The two curves below the 100% line are the total leakage achieved by our methods normalized to the best over 10K random search. The average leakage reductions are **10%** (leak_gr) and **24%** (leak_dc). The maximal leakage reductions are 46.4% and 60%, respectively. Similarly, the last two curves on top report the normalized circuit area, where we see little area increase (average **2%** for area_gr and **7%** for area_dc).

The run time for the fast gate replacement technique increases linearly to the number of gates in the circuit with an average of

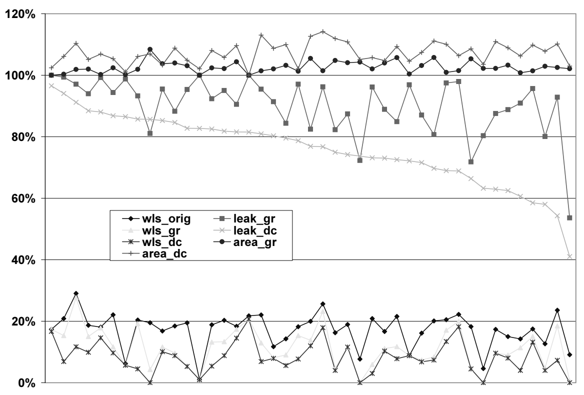

Figure 4: Results on 43 large circuits with 22 PIs or more.

0.05s. The average run time for the random search over 10K input vectors is **80.9s** and increases exponentially to the number of primary input and linearly to the number of gates. Excluding two circuits (*i8* and *des*) that has large tree circuits, the average run time for the divide-and-conquer algorithm is **142.55** and is less sensitive to the number of primary input.

Finally, we compare our results with those reported in [1]. Because their detailed results are not available and to make a fair comparison, we can only compare the average performance and leakage reduction over the average leakage current of 1,000 random vectors. Table 2 summerizes the performance improvement in the control point insertion approach [1], our gate replacement algorithm, and the divide-and-conquer approach.

Table 2: Average performance comparison with [1]

	algorithm in [1]	gate replacement	divide-and-conquer
leakage reduction	25%	23%	37%
delay penalty	$\leq 5\%$	0%	$\leq 5\%$
area penalty	$\leq 15\%$	2%	7%

5. CONCLUSIONS

We study the MLV+ problem which targets to reduce the number of gates at WLS for leakage reduction. We propose low-complexity heuristics to solve this problem. The proposed algorithms are practical and effective. The experimental results show that this is a promising direction for leakage reduction at gate level with little re-design overhead.

6. REFERENCES

[1] A. Abdollahi, F. Fallah, and M. Pedram, "Leakage Current Reduction in CMOS VLSI Circuits by Input Vector Control", *IEEE Trans. on VLSI*, Vol. 12, pp. 140-154, Feb. 2004.

[2] F. Aloul, S. Hassoun, K. Sakallah, D. Blaauw, "Robust SAT-Based Search Algorithm for Leakage Power Reduction", *International Workshop on Integrated Circuit Design*, pp. 167-177, 2002.

[3] B.H. Calhoun, F.A. Honore, and A. Chandrakasan, "Design Methodology for Fine-Grained Leakage Control in MTCMOS", *ISLPED*, pp. 104-109, 2003.

[4] D. Duarte, Y. Tsai, N. Vijaykrishnan, and M. Irwin, "Evaluating Run-Time Techniques for Leakage Power Reduction", *IEEE International Conference on VLSI Design*, pp. 31-38, 2002.

[5] F. Gao and J.P. Hayes, "Exact and Heuristic Approaches to Input Vector Control for Leakage Power Reduction", *ICCAD*, pp. 527-532, 2004.

[6] J. Halter, and F. Najm, "A Gate-Level Leakage Power Reduction Method for Ultra Low Power CMOS Circuits", *CICC*, pp 475-478, 1997.

[7] M.C. Johnson, D. Somasekhar, and K. Roy, "Models and Algorithms for Bounds on Leakage in CMOS Circuits", *IEEE Trans. on CAD*, Vol. 18, pp. 714-725, 1999.

[8] J. Kao, S. Narendra, A. Chandrakasan, "Subthreshold Leakage Modeling and Reduction Techniques", *ICCAD*, pp. 141-148, 2002

[9] R.M. Rao, F. Liu, J.L. Burns, and R.B. Brown, "A Heuristic to Determine Low Leakage Sleep State Vectors for CMOS Combinational Circuits", *ICCAD*, pp. 689-692, 2003.