A Combined Gate Replacement and Input Vector Control Approach for Leakage Current Reduction *

Lin Yuan and Gang Qu
Electrical and Computer Engineering Department and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742, USA

Abstract

Due to the increasing role of leakage power in CMOS circuit's total power dissipation, leakage reduction has attracted a lot of attention recently. Input vector control (IVC) takes advantage of the transistor stack effect to apply the minimum leakage vector (MLV) to the primary inputs of the circuit during the standby mode. However, IVC techniques become less effective for circuits of large logic depth because the MLV at primary inputs has little impact on internal gates at high logic level. In this paper, we propose a technique to overcome this limitation by directly controlling the inputs to the internal gates that are in their worst leakage states. Specifically, we propose a gate replacement technique that replaces such gates by other library gates while maintaining the circuit's correct functionality at the active mode. This modification of the circuit does not require changes of the design flow, but it opens the door for further leakage reduction, when the MLV is not effective. We then describe a divide-and-conquer approach that combines the gate replacement and input vector control techniques. It integrates an algorithm that finds the optimal MLV for tree circuits, a fast gate replacement heuristic, and a genetic algorithm that connects the tree circuits. We have conducted experiments on all the MCNC91 benchmark circuits. The results reveal that 1) the gate replacement technique itself can provide 10% more leakage current reduction over the best known IVC methods with no delay penalty and little area increase; 2) the divide-and-conquer approach outperforms the best pure IVC method by 24% and the existing control point insertion method by 12%; 3) when we obtain the optimal MLV for small circuits from exhaustive search, the proposed gate replacement alone can still reduce leakage current by 13% while the divide-and-conquer approach reduces 17%.

*Parts of this manuscript will appear in the 42nd ACM/IEEE Design Automation Conference.

1 Introduction

As the VLSI technology and supply/threshold voltage continue scaling down, 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 [15]. Many techniques thus have been proposed recently to reduce the leakage power consumption. Dual threshold voltage process uses devices with higher threshold voltage along non-critical paths to reduce leakage current while maintaining the performance [22]. Multiple-threshold CMOS (MTCMOS) technique places a high V_{th} device in series with low V_{th} circuitry, creating a sleep transistor [17, 4]. Calhoun et al. proposed a methodology to insert sleep transistors in MTCMOS [8]. In dynamic threshold MOS (DTMOS) [5], the gate and body are tied together and the threshold voltage is altered dynamically to suit the operating state of the circuit. Another technique to dynamically adjust threshold voltages is the variable threshold CMOS(VTCMOS) [19]. Controlling the body bias voltage to minimize leakage is discussed in [18]. All of these approaches require the process technology support.

(a)

INPUT	Leakage(nA)
0	best:100.3
1	worst:**227.2**

(b)

INPUT	Leakage(nA)
00	best: 37.84
01	2nd worst: 100.30
10	95.17
11	worst: **454.50**

(c)

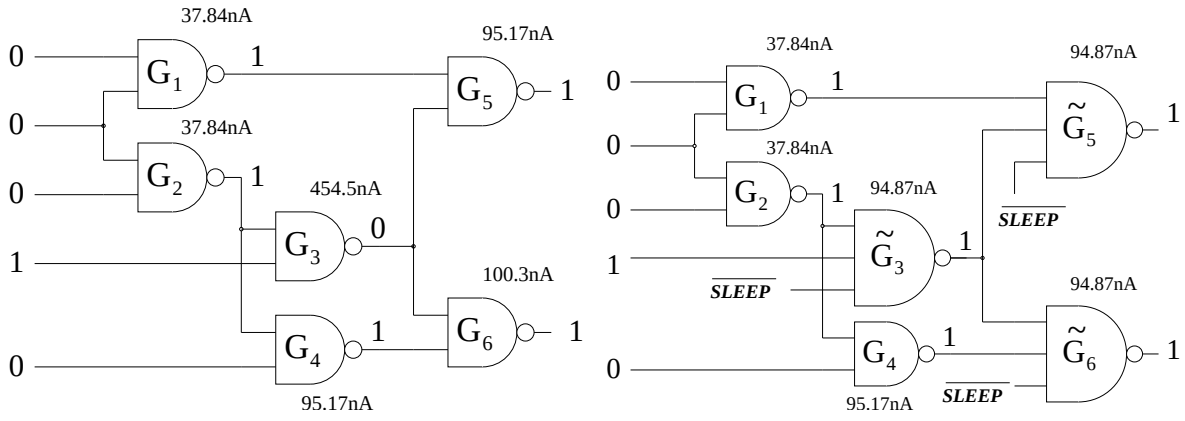
INPUT	Leakage (nA)
000	best: 22.84
001	37.84
010	37.84
011	2nd worst: 100.30
100	37.01
101	95.17
110	94.87
111	worst: **852.40**

Figure 1: Leakage current of (a)INVERTER, (b)NAND2 and (c)NAND3. Data obtained by simulation in Cadence Spectre using 0.18 μm process.

The input vector control (IVC) technique is applied to reduce leakage current at circuit level with little or no performance overhead [11]. It is based on the well-known transistor stack effect: a CMOS gate's subthreshold leakage current varies dramatically with the input vector applied to the gate [14]. Recently, Lee et al. made the similar observations on gate oxide leakage that it is also dependent on the input vectors to a CMOS gate [16]. In our study, we use Cadence Spectre to measure the overall leakage current in a CMOS gate that includes both subthreshold leakage and gate leakage. Figure 1 lists the overall leakage current in INVERTER, NAND2 and

NAND3 gates under all the possible input combinations. We see that the worst case leakage (marked in bold) is much higher than the other cases. The idea of IVC technique is to manipulate the input vector with the help of a sleep signal to reduce the leakage when the circuit is at the standby mode [13]. The associated minimum leakage vector (MLV) problem seeks to find a primary input vector that minimizes the total leakage current in a given circuit. [1, 6, 8, 10, 12, 13, 14, 20]. The MLV problem is NP-complete [1] and both exact and heuristic approaches have been proposed to search for the MLV. A detailed survey is given in Section 2.

In this paper, we consider how to enhance IVC technique with little or no re-design effort. In particular, we study the **MLV+ problem** that seeks to modify a given circuit and determine an input vector such that the circuit's functionality is maintained at the active mode and the circuit leakage is minimized when the circuit is at standby mode. Our solution to this problem is based on the concept of gate replacement that is motivated by the large discrepancy between the worst leakage and the other cases (see Figure 1). The essence of gate replacement is to replace a logic gate that is at its worst leakage state by another library gate. This is illustrated by the following example.



(a) Original MCNC benchmark circuit C17 with total leakage $831.08nA$ under the optimal MLV.

(b) New circuit C17 with three gates replaced and total leakage $476.88nA$ under the same MLV.

Figure 2: A motivation example for gate replacement.

Consider circuit C17 from the MCNC91 benchmark suite [26] (Figure 2(a)). An exhaustive search finds the MLV $\{0,0,0,1,0\}$, with the corresponding minimum total leakage current of $831.08nA$. Note that gate G_3 has its worst leakage current (454.5nA) with input $\{1,1\}$, which contributes more than half of the total leakage. In fact,

[1]The NP-completeness of the MLV problem has been mentioned by several research groups [14, 10, 12]; however, none of them gave a proof. In Section 4, we give one to make it complete.

we have observed that a significant portion of the total leakage is often caused by the gates that are in their worst leakage state (see Table 2 in Section 5).

Instead of controlling the primary inputs, we consider replacing these leakage-intensive gates. In particular, we replace the NAND2 gate G_3 by a NAND3 $\tilde{G}_3$ where the third input $\overline{SLEEP}$ is the complement of the $SLEEP$ signal (Figure 2(b)). At active mode, $\overline{SLEEP} = 1$ and $\tilde{G}_3$ produces the same output as G_3. But at the standby mode, $\overline{SLEEP} = 0$ and $\tilde{G}_3$ has a leakage of $94.87nA$ (Figure 1(b)), which is much smaller than G_3's $454.5nA$.

However, this replacement also changes the output of this gate at the sleep mode and affects the leakage on gates G_5 and G_6. In this case, we replace them in a similar fashion. As a result, the new circuit's total leakage becomes $476.88nA$, a 43% reduction from the original $831.08nA$ in Figure 2(a).

The proposed gate replacement technique is conceptually different from the existing input vector control methods. These two methods are complementary to each other. Specifically, IVC method considers the entire circuit and searches for an appropriate input vector in favor of small leakage. The gate replacement technique targets directly at the logic gates that are in their worst leakage state (WLS) under a specific input vector and replace them to reduce leakage. This paper has the following contributions:

1. Examination of the pure IVC methods[2]: For all the 69 MCNC91 benchmarks, we obtain the optimal MLV in small circuits with 22 or fewer primary inputs by exhaustive search; and the best over 10,000 random input vectors for large circuits. The number of gates in their WLS are on average 15% and 17% respectively, but they contribute more than 40% of the circuit's total leakage.

2. Gate replacement for leakage reduction: Our work is motivated by the above observation. The basic idea is to replace gates that are in their WLS by other library gates that will generate less leakage current at those states. Unlike other leakage reduction techniques such as MTCMOS and DTMOS, this modification of the circuit does not require changes of process technology in the design flow. Therefore, it will not increase the design complexity or the leakage sensitivity.

3. A fast gate replacement technique: We implement a simple gate replacement technique that gives an average of 10% leakage reduction for a fixed input vector. If we apply it to the optimal/sub-optimal MLV mentioned above in 1, the number of gates in their WLS is reduced to 8% and 11%, respectively. This algorithm's run time complexity is linear to the number of gates in the circuit in average cases and quadratic in the worst case.

[2]IVC-based approaches such as internal control point insertion [1] will be discussed in Section 2

4. Solving the MLV+ problem: We develop a divide-and-conquer approach to combine gate replacement and IVC. It reduces the leakage by 17% and 24% over the optimal/sub-optimal MLV mentioned in 1) with little area and delay overhead. The number of gates in their WLS is dropped to 4% and 9% respectively.

The rest of the paper is organized as follows: in the next section, we review the IVC leakage reduction techniques. In Section 3, we describe the proposed gate replacement technique. In Section 4, we elaborate the proposed divide-and-conquer approach. Detailed experiment results on all 69 MCNC91 benchmarks are reported in Section 5 before we conclude in Section 6.

2 Input Vector Control for Leakage Current Reduction

Reducing the off-state leakage power/current has become a primary concern for low power circuit design recently. We have already mentioned a number of techniques proposed to reduce leakage in the introduction. More detailed review and survey can be found in [4, 7, 11]. In this section, we survey the efforts on the input vector control (IVC) techniques proposed for finding the minimum leakage vector.

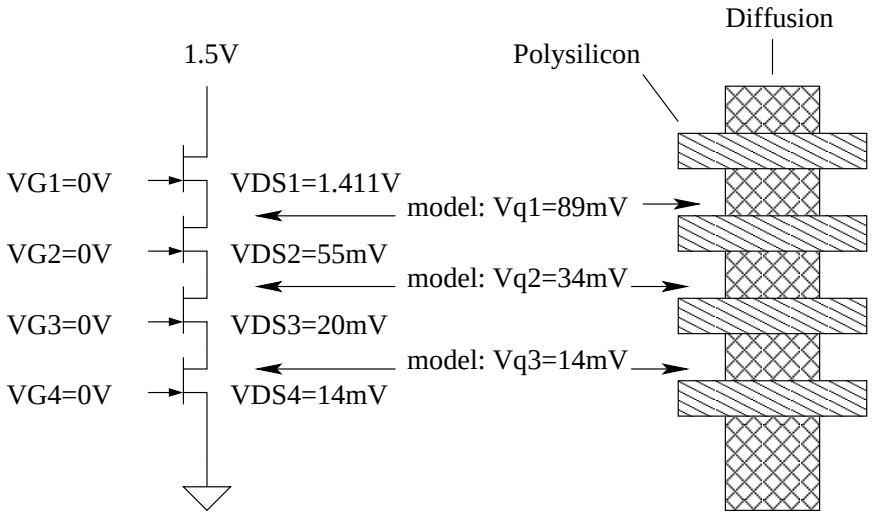


Figure 3: Schematic layout and typical quiescent voltages of four transistor stack[14]

The effect of circuit input logic values on leakage current was observed by Halter and Najm [13]. The underlying reason of this effect was explained by Johnson et al. [14]. They built a model to calculate leakage current in a stack of transistors as shown in Figure 3. In this model, transistors which have a gate voltage equal to V_{DD} are treated as short circuits and voltage drop across transistors that are off is calculated. Their model infers that, the more transistors being turned off in a stack of transistors, the larger the effective resistance these

transistors will have, and the less leakage currents pass through them. This effect is called transistor stack effect. Hence, the IVC technique is essentially to control the number of off transistors in the circuit. Authors in [13, 23] proposed a technique to insert a set of latches with MLV stored in to the primary inputs of a circuit, forcing the combinational logic into a low-leakage state when the circuit is idle. In [10], the authors also discussed the trade off between the optimality of the minimum leakage state and the switching cost of entering and exiting this state.

Apparently, IVC method provides the maximal leakage reduction when the input vector gives the minimal total leakage. Besides, it does not require process technology modification and can be applied at runtime. Many algorithms have been proposed to find such minimum leakage vectors (MLV). Based on the nature of these algorithms, we classify them into three groups: **heuristics, exact algorithms**, and **internal point control**.

2.1 Heuristic Algorithms

In [13], Halter and Najm developed a random sampling-based heuristic to find the MLV. They started with a set of identically and independently distributed random vectors. These vectors are applied to the circuit as primary inputs and the one that gives the least leakage current is selected. They further calculated *error tolerance*: the probability that another random vector results in leakage less than the selected. Their experiments showed that the MLV selected from 100,000 random vectors can realize an error tolerance of 1% or smaller with a 54% leakage reduction over the vector that generates the largest leakage.

Chen et al. [9] proposed a genetic algorithm to tackle the MLV problem, which has a solution space exponential to the number of primary inputs. In their genetic algorithm, an input vector is represented as a chromosome and the circuit's total leakage is calculated and used as the fitness of the chromosome. They select a random population and produce the next $generation$ by standard genetic algorithm operations such as $selection$, $crossover$, and $mutation$. When the stopping criterion is satisfied, the best overall solution is reported as the MLV. In additional to the minimal leakage, their method can also find the maximal leakage vector. The results show that the ratio of maximal leakage to minimal leakage can be as large as 5, which indicates that the IVC technique have great potentials in reducing leakage power.

Johnson et al. [14] explained the transistor stack effect in CMOS circuits and developed analytical models to estimate the steady-state leakage and the duration of leakage transients in series connections of MOS transistors. Based on these models, they defined *leakage observability* as the degree to which the value of a particular circuit input is observable in the magnitude of leakage from the power supply. The leakage observability of each primary input is evaluated and the primary inputs are put into a priority queue with their leakage observability as the priority metric. For the primary input on top of the priority queue, assign it a value 0 or 1 based on which one

gives a smaller leakage; the circuit state and leakage observability are updated under this new input value; the priority queue is then updated and the next primary input on top of the queue will be assigned a value until the queue becomes empty. The input combination constructed in this greedy fashion is taken as the MLV.

In [20], Rao et al. proposed a fast heuristic algorithm based on the concept of controllability that is widely-used for fault detection in automatic test pattern generation. The controllability of each node in the circuit is defined as the minimum number of inputs that need to be assigned to particular states in order to force the given node to a specific state. A cost function for each node is also defined as the difference between the node's best case (smallest) leakage and worst case (largest) leakage. In their heuristic, the controllability is first computed for each internal node (gate) in the circuit. The node with the least cost function and satisfies its input constraint is then removed. The controllability list is updated and this process is repeated until all the nodes are removed from the list. If there exist input undefined at the end, the value that results in smaller leakage is assigned. Their experimental results showed a leakage within 5% of the best vector obtained from 100,000 random vectors, but with a large run time saving.

2.2 Exact Algorithms

Several exact algorithms have been proposed to find the MLV recently. These include a graph-based Boolean enumeration method [10], a pseudo Boolean Satisfiability formulation [1, 2], and an integer linear programming formulation [12]. They transform the MLV problem into other well-studied problems (such as ILP) and solve them by the existing solvers.

The MLV problem is modeled as a pseudo Boolean Satisfiability (PBS) problem in [1, 2]. There are two sets of constraints in the PBS problem formulation: the constraints that represent the circuit's functionality and those that represent the leakage objective. As an example, consider a 2-input NAND gate. Given the leakage power values listed in Figure 1, the leakage objective constraints will be expressed as:

$$37.84\bar{X}_1\bar{X}_2 + 100.3\bar{X}_1X_2 + 95.17X_1\bar{X}_2 + 454.5X_1X_2 \leq k$$

where X_1, X_2 are inputs to the gate; k is the desired leakage bound. A minimal leakage can by found by iteratively reducing k until the formula is unsatisfiable.

In order to have the product-of-sum format, X_1X_2 term is replaced by a variable S with the help of logic constraints:

$$(\bar{X}_1 + \bar{X}_2 + S) \cdot (X_1 + \bar{S}) \cdot (X_2 + \bar{S})$$

The SAT formula of a circuit is the AND of all the constraints associated with each gate. Once this PBS

problem is established, advanced solver PBS [3] can be used to find the MLV for leakage reduction. In [2], they can improve the leakage by up to 12%. However, as the circuit size increases, the runtime of this approach rises dramatically. In large circuits such as C1355 and C6288, their approach took more than 5000 seconds CPU time and didn't complete for this reason.

Gao and Hayes [12] formulate the MLV problem as an integer linear programming(ILP) problem. They first use pseudo-Boolean functions to represent leakage current in different types of cells with the general sum-of-products form:

$$F(i_0, i_1, ..., i_{n-1}) = \sum_{C \in \Re} C \prod_{0 \le j \le n-1} i_j$$

For example, the leakage in a two-input NAND gate G can be expressed as

$$F(i_0, i_1) = L_{00}\bar{i_0}\bar{i_1} + L_{10}i_0\bar{i_1} + L_{01}\bar{i_0}i_1 + L_{11}i_0i_1$$

where $F(i_0, i_1)$ is the leakage current in G with input vector i_0i_1 and L_{ab} denotes G's leakage current with input vector $i_0i_1 = \{a, b\}$. Applying the well-known Boole-Shannon expansion [25], this equation can be transformed to

$$F(i_0, i_1) = F(0,0) + C_0i_0 + C_1i_1 + C_2i_0i_1$$

where $C_0 = F(1,0) - F(0,0)$ and $C_1 = F(0,1) - F(0,0)$ are the coefficients for terms with one input variable. Similarly, the coefficients of the two-variable term is $C_2 = F(1,1) - F(1,0) - F(0,1) + F(0,0)$. Then the authors use an extra variable X for term i_0i_1, the new function is then linear in i_0, i_1 and X. This transformation can be generalized to linearize the leakage function for an arbitrary gate with n inputs:

$$F(i_0, i_1, ..., i_{n-1}) = \sum_{S} C(S)X(S)$$

where S, an auxiliary variable, is a subset of input variables; $X(S)$ is the logic AND of all variables in S; $C(S)$ is the coefficients defined as:

$$C(S) = \sum_{All\ subsets\ T\ of\ S} (-1)^{|S|-|T|}F(T).$$

Additional constraints are also formed using linear equations or inequations to guarantee the correct logic relations in the circuit. For example, the functionality of an n-input NAND gate is represented by:

$$\begin{aligned} f &\le 2 - (i_1 + i_2 + ... + i_n + 1)/n \\ f &\ge 1 - (i_1 + i_2 + ... + i_n)/n \end{aligned}$$

where $i_1, i_2...i_n$ are inputs to NAND gate; f is the output.

After the ILP model is built, an off-the-shelf ILP solver is used to obtain the MLV. For large circuits, the size of the ILP formula could be too large for a solver. The authors proposed a simplified mixed-integer linear programming formulation that uses selective variable-type relaxation to reduce the runtime. Their experiments on a set of benchmark circuits showed that the latter relaxation can speedup the ILP approach by 13.64X with 4% error.

Based on the pseudo Boolean function formulation of the leakage in CMOS gates, two implicit pseudo boolean enumeration algorithms are presented in [10]. The input space enumeration method leverages integer valued decision diagrams and works well for small circuits. The hypergraph partitioning based recursive algorithm represents a given circuit as a hypergraph where the vertices of the hypergraph are gates and the hyperedges are nets. It partitions a large circuit into subgraphs using Min-Cut algorithm and solves independently for each subgraph. Their algorithms can find the optimal MLV much faster than other exact algorithms. The authors also discussed how to reduce the number of primary inputs that must have a specific value in the MLV in order to save the switching power for the circuit to enter the MLV state.

2.3 Internal Point Control

When a circuit has many logic levels, the IVC technique becomes less effective because the internal gates at a deep level are less affected by the primary input vectors. For this reason, Abdollahi et al. proposed a technique to control the value of internal pins to reduce leakage [1]. Their first approach inserts multiplexers at the input pins of each gate. The $SLEEP$ signal selects the correct input in active mode and chooses the input values that produce low leakage current in standby mode. This approach can reduce leakage in the CMOS gates significantly; however, the inserted multiplexers will also generate leakage current and introduce extra delay and area. To compensate this overhead, the authors formulate the insertion of multiplexers using pseudo boolean SAT. Again, since the solution place to insert multiplexer is enormous, the runtime of PBS becomes unmanageable for large circuits.

In their second approach, they modify the library gates by adding $SLEEP$ signal-controlled transistors in the gate to select the low-leakage inputs for its fanout gates. They reported an average leakage reduction of 25% within 5% delay penalty and no more than 15% area increase. However, since the structure of the gates is changed, a new set of library gates are needed.

Our gate replacement technique belongs to the class of internal point control, but is conceptually different from [1] in the following aspects: 1) They treat each input pin of the gates as potential place to insert multiplexers, while we consider only roots of each tree. The search space is reduced substantially. 2) Their purpose of

modifying a gate G is to produce the low-leakage input for G's fanout gate while we aim to reduce leakage current at G itself. 3) They modify gates whenever necessary while we restrict our algorithm to replace gates only by the available gates in the library, and hence do not require gate structure modification. However, these two approaches can be combined as we will discuss in more details in Sections 3 and 4.

3 Leakage Reduction by Gate Replacement

A logic gate is at its worst leakage state (WLS) when its input yields the largest leakage current. Regardless of the primary input vector, a large number of gates are at WLS, particularly when the circuit has high logic depth. Take the 69 MCNC91 benchmarks for example. For each of the 69 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 5. In this section, we describe the gate replacement technique that targets directly the leakage reduction in WLS gates.

3.1 Basic Gate Replacement Technique

As we have shown in the motivation example in Section 1, the proposed gate replacement technique replaces a gate $G(\vec{x})$ by another library 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 smaller leakage than $G(\vec{x})$ when the circuit is in standby ($SLEEP = 1$).

The first condition guarantees the correct functionality of the circuit at active mode. The second condition reduces the leakage on gate G at the standby mode, but it may change the output of this gate. Note that, 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.

Figure 4(a) shows that the replacement of G by $\tilde{G}$ changes the output from 0 to 1. For simplicity, we assume that G's fanout only goes to gate H which can be either a NAND or a NOR or an INVERTER. In Figure 4(b) and (d), we see that such change does not affect the output of gate H and therefore it won't affect any other gates in the circuit. Let $L(G(11))$ be the leakage of gate G with input 11, we can conveniently compute the leakage reduction by this replacement, which is $L(G(11)) + L(H(00)) - L(\tilde{G}(110)) - L(H(10))$ in the case of (b) for example.

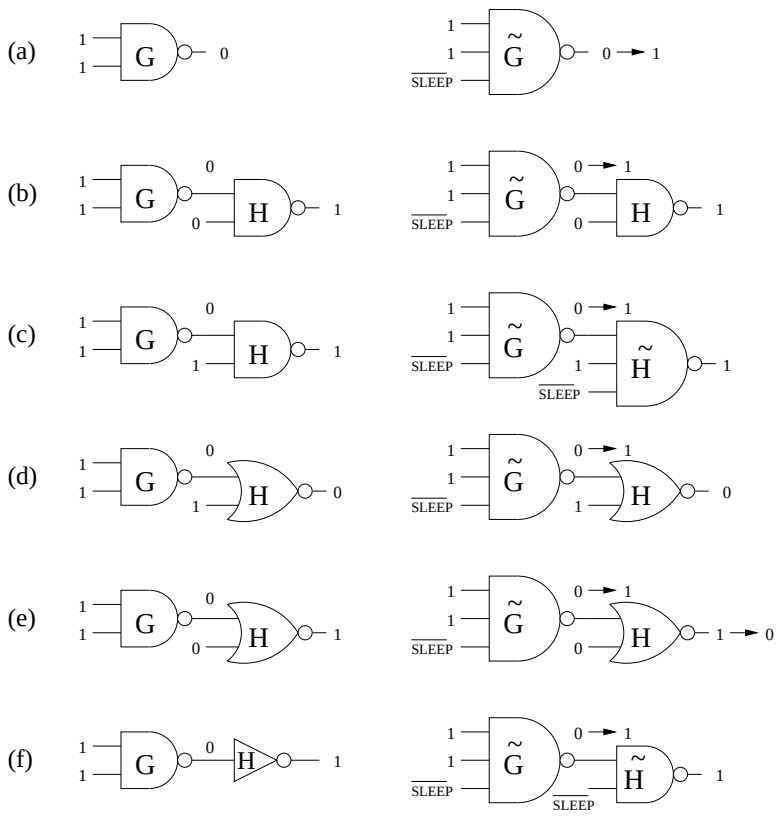


Figure 4: Gate replacement and the consequence to its fanout gate.

In Figure 4(c), the replacement at gate G not only changes the output of gate H, it also puts H at its WLS. Our solution is to replace the NAND2 gate H by an NAND3 $\tilde{H}$. This preserves the output of H and the leakage change will be $L(G(11)) + L(H(01)) - L(\tilde{G}(110)) - L(\tilde{H}(110))$. Similarly, in Figure 4(f), we replace the INVERTER by a NAND2 gate. Finally, in Figure 4(e), the replacement of G moves both gates G and H away from their WLS. It also changes the output of the NOR gate H, which we can conduct similar analysis.

Remarks:

General Fanout. The above analysis is applicable to G's fanout gate H of any type. The change of G's output either does not affect H's output (Figure 4 (b) and (d)) or changes H's output. In the latter case, we either change H's output back (Figure 4 (c) and (f)) or continue the analysis starting from H (Figure 4 (e)).

Beyond library gates. If the library does not have a replacement for G, we can add one transistor into the N or P sections of G to meet conditions 1 and 2. This is similar to the gate modification method proposed in [1]. However, they attempt to control the output of the modified gate in order to reduce the leakage in its fanout gate by producing the desirable signal. Our gate replacement targets directly the leakage reduction

of the current gate.

Multiple fanouts. When gate G has multiple fanouts, we analyze each of them and then consider their total leakage when we compute the leakage change due to the replacement of gate G.

Compatibility. The gate replacement technique does not change the primary input vector of the circuit. This implies that we can combine it with existing MLV searching strategies to further reduce leakage. The MLV+ problem is based on this observation and is discussed in details in next section.

Power overhead. There is not much dynamic power overhead because the SLEEP signal remains constant at active mode and will not cause any additional switching activities. The leakage in gates $\tilde{G}$ and G may be different at active mode. Such difference becomes negligible when the circuit stays at standby mode long enough [1].

Other overhead. Gate replacement may introduce delay and area overhead. This overhead can be controlled by restricting the replacement off critical path and transistor resizing. Gate replacement does not add new logic gates and thus requires little or no effort to redo the place-and-route.

3.2 A Fast Gate Replacement Algorithm

Based on the above gate replacement technique, we propose a fast algorithm that selectively replaces gates to reduce the circuit's total leakage for a given input vector. Figure 5 gives the pseudo-code of this algorithm.

We visit the gates in the circuit by the topological order. We skip all the gates that are not at WLS and the gates that have already been visited or marked (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 and other gates in $\mathcal{S}$. 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.

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 $\mathcal{S}$

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 ∈ {G_1, G_2, ···}
2.   if (G_i is at WLS and not marked)
3.     include G_i in the selection S;
4.     while (there is new addition to S)
5.       for each newly selected gate G in S
6.         if (there exists library gate G̃ meets the conditions in Section 3.1)
7.           temporarily replace G by G̃;
8.           if (output of G is changed due to this replacement)
9.             include G's unmarked fanout gate G_j in S;
10.    compute the total leakage change of gates in S;
11.    if (there is leakage reduction)
12.      mark all gates G_j in the selection S;
13.      make the replacements in lines 7,9,or 10 permanent;
14.    else mark gate G_i only;
15.    empty the selection S;
16.  else mark G_i if it has not been marked yet;
```

Figure 5: Pseudo-code of the gate replacement algorithm.

(line 10). 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 and the marking of all gates in $\mathcal{S}$ (line 10-15) 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 4), 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 4 (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.

Improvement: There are several ways to improve the leakage reduction performance of the above gate replacement heuristic. The tradeoff will be either increased design complexity, or reduced circuit performance, or both. First, one can consider gates that are not in the library as we have commented in the remarks in Section 3.1 (line 6). However, this requires the measurement of leakage current, area and delay in these new gates as they are not available in the library. A second alternative is to insert control point at one of G's fanins. For example, one can find the fanin y such that replacing y by its complement y' gives G the largest leakage reduction. If $y = 0$, replace it by $OR(y, SLEEP)$; if $y = 1$, replace it by $AND(y, \overline{SLEEP})$. However, the addition of new gates may require the repeat of placement and routing and will incur more area and delay penalty in general. Third, one may also consider both the library gate replacement and control point insertion at the same time and choose the one that gives more leakage reduction. Finally, whenever we replace gate G_i, we also make the replacement for all the other gates in the selection $\mathcal{S}$ permanent (line 13). We have tested a couple of alternatives and they give limited improvement in leakage reduction at very high cost of run time complexity.

The incentive to keep the run time complexity of this gate replacement algorithm low is that it will be combined with IVC technique under the following divide-and-conquer approach to solve the MLV+ problem.

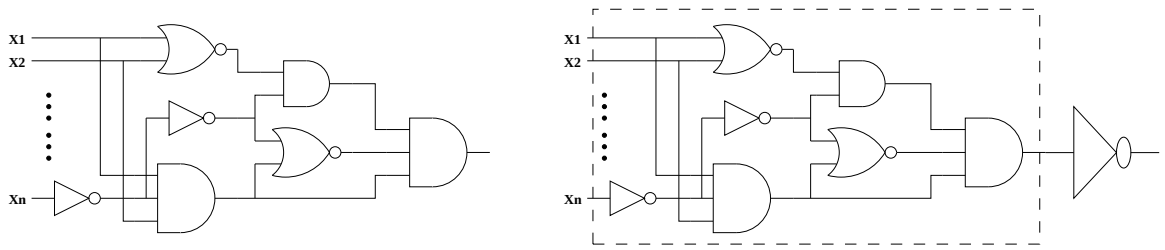
4 The MLV+ Problem and the Divide-and-Conquer Approach

Recall that the minimum leakage vector (MLV) problem seeks for the input vector that minimizes the circuit's total leakage. It has been claimed that this problem is NP-complete for general circuits [1, 10, 14, 20]. But no formal proof has been given to our knowledge. In this section, we first give a brief proof of the NP-completeness of the MLV problem and then define the MLV+ problem, an extension of the MLV problem. Our main focus will be on the divide-and-conquer approach that solves the MLV+ problem.

4.0 NP-Completeness of the MLV Problem

The MLV problem could be stated as follows: given a combinational circuit consisting of primary inputs (PIs), primary outputs(POs), internal logic gates connected by nets/wires, and the leakage current of each gate under different input combinations, determine an input vector at the PIs such that the total leakage current of all the gates in the circuit is minimized.

Theorem: The MLV problem is NP-complete.



(a) A circuit for satisfiability test. (b) Reducing the satisfiability test to MLV.

Figure 6: Illustration for the proof of the NP-completeness of the MLV problem.

Proof. On one side, we have already mentioned a couple of exact algorithms that solve the MLV problem by reducing it to NP-complete problems such as pseudo Boolean satisfiability and integer linear programming.

On the other side, we show that the NP-complete CIRCUIT-SAT problem [24] can be reduced to the MLV problem. Consider an arbitrary circuit shown in Figure 6(a), to test whether the circuit is satisfiable (i.e., producing a logic '1' at its output), we construct a new circuit by adding a big inverter at its output (Figure 6(b)). The inverter is big in the sense that it has a huge leakage value L when its input is '0' and a small leakage ϵ when its input is '1'. Actually, we can set L to be the sum of ϵ and the leakage of each gate in the circuit when it is in its WLS. Now we solve the MLV problem for this modified circuit. If the total leakage is less than L, clearly the original circuit is satisfiable and the MLV is one input vector that makes the circuit output logic '1'. Otherwise, because that the only way for the total leakage to be larger than L is when the input to the big inverter is '0', the original circuit is not satisfiable. □

4.1 The MLV+ Problem and Outline of the Divide-and-Conquer Approach

Note that the MLV problem seeks for the input vector to a circuit that minimizes the circuit's total leakage. In the previous section, we have seen that leakage current can be further reduced by the proposed gate replacement technique. We have also mentioned that this technique is independent of the input vector and can be combined with the MLV method. We hence formulate the following **MLV+ problem**:

> *Given a combinational circuit with PIs, POs, the internal logic gates that implement the PI-PO functionality, and the leakage current of each library gate under its different input patterns, determine a gate level implementation of the same PI-PO functionality without changing the place-and-route and an input vector at the PIs that minimizes the total leakage.*

Apparently , this is an extension of the MLV problem with the relaxation of modifying circuit by gate replacement. It enlarges the search space of MLV and provides us with the opportunity of finding better solution.

For a circuit of k PIs and n internal logic gates, the search space for the original MLV problem is the 2^k different input combinations. Under the above MLV+ formulation, the search space becomes $2^k \cdot \Pi_{i=1}^{n} l_i$, where l_i is the number of library gates that can replace gate i, including gate i itself. Assuming that half of the gates have one replacement, then the solution space for MLV+ problem will be $2^{n/2}$ times larger than the solution space for the MLV problem. Even when we restrict the gate replacement technique only to gates that are at their WLS, this will be significant because (1) a circuit normally has more gates than PIs ($n >> k$) and (2) the percentage of gates in WLS is considerably high (16% on the MCNC91 benchmark when MLV is applied, and will be higher as the logic depth of the circuit increases).

As we have analyzed in the previous section, the MLV+ problem not only enlarges the solution space for the IVC method, it also has the great potential in improving the solution quality (in terms of leakage reduction) because of the stack effect. However, one challenge is how to explore such enormous solution space for better solutions. Given the NP-completeness of the MLV problem, we consider special circuits where the MLV+ can be solved optimally and develop heuristics for the general case. In the rest of this section, we describe details of our proposed divide-and-conquer approach that consists of the following phases:

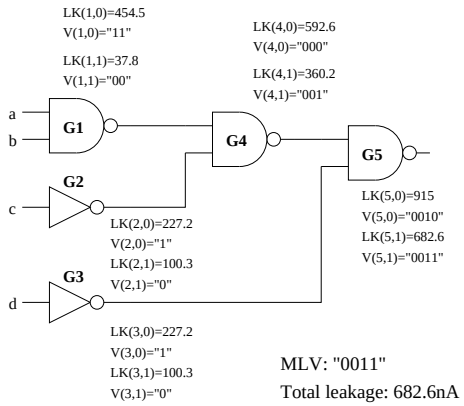
1. decompose a general circuit into tree circuits.
2. find the MLV for each tree circuit optimally by dynamic programming.
3. apply the gate replacement technique to the MLV for each tree to further reduce leakage.
4. connect the tree circuits by a genetic algorithm.

4.2 Finding the Optimal MLV for Tree Circuits

A tree circuit is a single output circuit in which each gate, except the primary output, feeds exactly one other gate. A general combinational circuit can be trivially decomposed into non-overlapping tree circuits [25]. This is illustrated in Figure 9. The circuit in (a) is not a tree because gate G_3 has two fan-out gates G_5 and G_6. By splitting at the fanout of G_3, we get three trees with G_3, G_5 and G_6 being the root of each tree respectively.

We consider a tree circuit with gates $\{G_1, G_2, \cdots, G_n\}$ sorted in the topological order, which is preserved by the tree decomposition.

Let $L(G_i(\vec{x}))$ be the leakage current in the gate G_i when vector $\vec{x}$ is applied at G_i's fanins. Each gate G_i can be treated as the root of a sub-tree circuit. Let $LK(i, z)$ be the minimum total leakage of the tree circuit when it outputs logic value z at root G_i and $\vec{V}(i, z)$ be the input vector to the tree circuit that achieves $LK(i, z)$. We develop a dynamic programming approach to compute the pairs $(LK(i, 0), \vec{V}(i, 0))$ and $(LK(i, 1), \vec{V}(i, 1))$



LK(1,0) = L(G1("11")) = 454.5 V(1, 0) = "11"
LK(1,1) = min(L(G1("11")), L(G1("10")), L(G1("00"))) = L(G1("00")) = 37.8
V(1, 1) = "00"

LK(2, 0) = L(G2("1")) = 227.2 V(2, 0) = "1"
LK(2, 1) = L(G2("0")) = 100.3 V(2, 1) = "0"

LK(3, 0) = L(G3("1")) = 227.2 V(3, 0) = "1"
LK(3, 1) = L(G3("0")) = 100.3 V(3, 1) = "0"

LK(4, 0) = L(G4("11")) + LK(1, 1) + LK(2,1)=592.6
V(4, 0) = "000"
LK(4, 1) = min{ L(G4("10"))+LK(1,1)+LK(2,0), L(G4("01"))+LK(1,0)+LK(2,1),
L(G4("00"))+LK(1,0)+LK(2,0)} = L(G4("10"))+LK(1,1)+LK(2,0)=360.2
V(4, 1) = "001"

LK(5, 0) = L(G5("11"))+LK(4,1)+LK(3,1) = 915
V(5, 0) = "0010"
LK(5, 1) = min{ L(G5("10"))+LK(4,1)+LK(3,0), L(G5("01"))+LK(4,0)+LK(3,1),
L(G5("00"))+LK(4,0)+LK(3,0)} = L(G5("10"))+LK(4,1)+LK(3,0) = 682.6
V(5, 1) = "0011"

Figure 7: Dynamic programming to find optimal MLV in a tree circuit.

for each gate G_i. The MLV for the tree circuit rooted at gate G_n, with gates $\{G_1, G_2, \cdots, G_n\}$ sorted in the topological order, can then be determined conveniently.

1. For each input signal to the tree, define

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

2. For each gate $G_i (i = 1, 2, ..., n)$, let

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

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

   where $\{x_{i_1}, x_{i_2}, \cdots, x_{i_t}\}$ are the fanins of G_i from gates $\{G_{i_1}, G_{i_2}, \cdots, G_{i_t}\}$ respectively and the input combination $\{x_{i_1}^0, \cdots, x_{i_t}^0\}$ achieves $LK(i, z)$.

3. The minimum leakage of the tree circuit with gates $\{G_1, \cdots, G_n\}$ is given by

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

   and the MLV will be either $\vec{V}(n, 0)$ or $\vec{V}(n, 1)$ accordingly.

Figure 7 gives a step-by-step illustration of the dynamic programming on a small circuit.

Correctness: We show the correctness of the recursive formula in Equation (2) and (3). 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 combination, 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 fan-in gate G_{i_j} with output x_{i_j}, $LK(i_j, x_{i_j})$. Equation (2) takes the overall minimum leakage and gives the correct $LK(i, z)$. Assume that this minimum leakage is achieved when G_i has fanins $x_{i_1} = x^0_{i_1}, ..., x_{i_t} = x^0_{i_t}$. Note that $\vec{V}(i_j, x^0_{i_j})$ is the input vector for the subtree circuit rooted at G_j to produce $x^0_{i_j}$ with the minimum leakage $LK(i_j, x_{i_j})$. The tree structure of the circuit guarantees that the subtrees rooted at $\{G_{i_1}, ..., G_{i_t}\}$ will not share any common inputs. Therefore, $\vec{V}(i, z)$ is the simple concatenation of $\vec{V}(i_j, x^0_{i_j})$ as given in Equation (3).

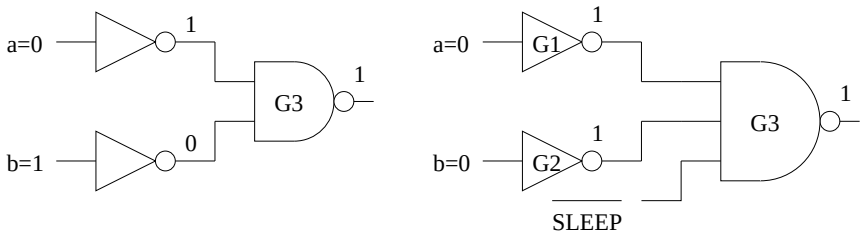
Complexity: Equations (1) and (4) take constant time. For each gate G_i, we need to compute $(LK(i, 0), \vec{V}(i, 0))$ and $(LK(i, 1), \vec{V}(i, 1))$ by equations (2) and (3). This requires the enumeration of all the 2^t different combinations of G_i's t fanins. For the first time, we need to perform t additions in equation (2). If we enumerate the rest $2^t - 1$ cases following a Gray code, we only need to update $L(G_i(\vec{x}))$(two operations), replace one $LK(i_j, x_{i_j})$ (two operations) 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.

After obtaining the MLV for the tree circuit, we perform the gate replacement algorithm proposed in Section 3 to further reduce leakage. Note that, although the MLV is optimal, this does not guarantee us an optimal solution for the MLV+ problem on the tree circuit. For example, consider the circuit in Figure 8, the algorithm finds the optimal MLV {a=0, b=1} with leakage 422nA. Gate 2 is at its WLS and the gate replacement algorithm does not give any improvement. The input vector {0,0} gives the maximum leakage 654nA; however, when we apply gate replacement technique and replace G_3, the leakage is reduced to 295nA. In fact, {0,0} is the optimal solution for the MLV+ problem. [3].

4.3 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 may have 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. For example, in Figure 9 (a), the circuit is decomposed into three tree

[3]We conjecture that the MLV+ problem remains NP-hard for tree circuit. Because we have already lost the optimality when we do the tree decomposition, we will not discuss in details on how to find better solutions to MLV+ on tree circuits. For the same reason, we did not focus on how to improve the fast gate replacement algorithm in Section 3.2



minimum leakage = LK(3,1)=422.6nA
MLV = "01"

minimum leakage = LK(3,1)=295.5nA
MLV = "00"

Figure 8: MLV in a circuit before and after gate replacement

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.

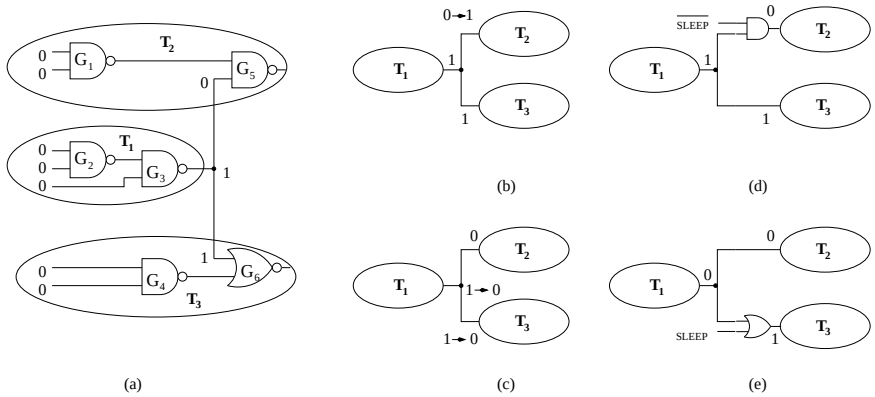


Figure 9: Resolving the conflict in connecting tree circuits.

There are basically three ways to resolve this conflict:

(I) enforcing T_1's output at all the fanout gates (Figure 9 (b));

(II) changing T_1's output and enforcing this new value at all the fanout gates (Figure 9 (c));

(III) inserting an AND gate to allow them to be inconsistent (Figure 9 (d)). Similarly, if T_1 output '0' and some of its fanouts require '1', we can add an OR gate as shown in Figure 9 (e)).

To decide which one we should use to resolve the conflict, we apply each of them and 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 logic '1'. The dynamic programming algorithm in Section 4.2 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-controlled 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 now easy 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 9 (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 4.2 to re-compute the minimum leakage of a tree circuit when methods (I) and (II) are used, run the gate replacement algorithm in Figure 5 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.

4.4 Overhead Analysis

As the control gates are introduced in the tree-connecting stage of the algorithm, they also require sleep signal to control. Hence, we need to consider the extra power these control gates and sleep signal may consume, and their effect on the overall power saving. In this subsection, we will discuss the power overheads.

1) Control gates: The control gates will consume extra dynamic power and leakage power. In this paper, we only consider the leakage power overhead of the inserted gates and ignore their dynamic power due to the following reasons. First, the number of inserted control gates only accounts for 5% to 6% of the total number of gates in the circuit. Second, they are simple 2-input AND and OR gates, which have a relatively small intrinsic capacitance at the node compared to other gates. Third, the switching activities in these control gates are very limited because one of the two inputs is the sleep signal, which changes only at the moment when the circuit switches between active mode and sleep mode. As dynamic power is dependent on physical capacitance and switching activities, we consider this dynamic power overhead is negligible.

As for leakage power, we measured the average leakage current in control gates over all possible inputs. In our algorithm, we add this extra leakage current to the objective function, i.e., the overall leakage current to be minimized. Therefore, the leakage saving achieved in our algorithm has already considered this overhead.

2) Sleep signal: Both the gate replacement and the control gates require the sleep signal to drive them during active and sleep mode. The generation of the sleep signal may consume extra power. However, due to the fact that our experiment was conducted at the logic synthesis level before placement and routing, it is not practical to obtain such power data quantitatively. On the other hand, the sleep signal is required by many other leakage minimization techniques, such as [1], [4], [5], [8] and [17]. Hence, in this paper, we expect the generation of the sleep signal to be similar to those approaches and we believe this problem can be better solved at the physical level of circuit design.

5 Experimental Results

We implemented the gate replacement and divide-and-conquer techniques in SIS environment [27] 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 measured leakage current includes both subthreshold and gate leakage. The simulations are conducted on a Ultra SPARC SUN workstation.

Our results are compared with traditional input vector control methods in terms of leakage saving, run time, area and delay penalty. The 69 benchmarks including 26 small circuits with 22 or fewer primary inputs (Table 1) and 43 large circuits (Table 2). For each small circuit, we find the optimal MLV by exhaustive search. For each large circuit, we choose the best MLV from 10,000 distinct random input vectors. It is reported that this will give

us a 99% confidence that the vectors with less leakage is less than 0.5% of the entire vector population [13, 20]. 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 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 move 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 off critical paths. This leaves 8% of the gates in the circuits at their WLS, but it also guarantees us that 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 10 reports the leakage and wls gates reduction in the 43 large circuits (x-axis) with 22 PIs or more. We replace the infeasible exhaustive search by the best solution from a random search of 10K input vectors. The fast gate replacement algorithm are restricted only on gates off critical paths; for the divide-and-conquer approach, we set the maximal delay increase to be 5%.

The benchmarks are sorted by the total leakage achieved by the divide-and-conquer method normalized to the best over 10K random search, which is shown one of the two curves at the top part of the figure. The average leakage reductions are 10% by gate replacement only (leakage G.R.) and 24% by divide-and-conquer method (leakage D.C.). The maximal leakage reductions are 46.4% and 60% respectively. The three curves at the bottom give the ratio of WLS gates. On average, the 10K random search has 17% gates at WLS(orig, wls); the proposed fast gate replacement and divide-and-conquer techniques reduce this ratio to 11%(G.R. wls) and 9%(D.C. wls), respectively.

More detailed results for these 43 circuits are shown in Table 2. Columns 4-6 list the leakage current, runtime, and percentage of gates at WLS when the best MLV from 10,000 random vectors is applied to each circuit. The next four columns show the results when the fast gate replacement algorithm is applied to such best MLV. The average run time is only 0.05s and increases linearly to the number of gates in the circuit. There is no delay

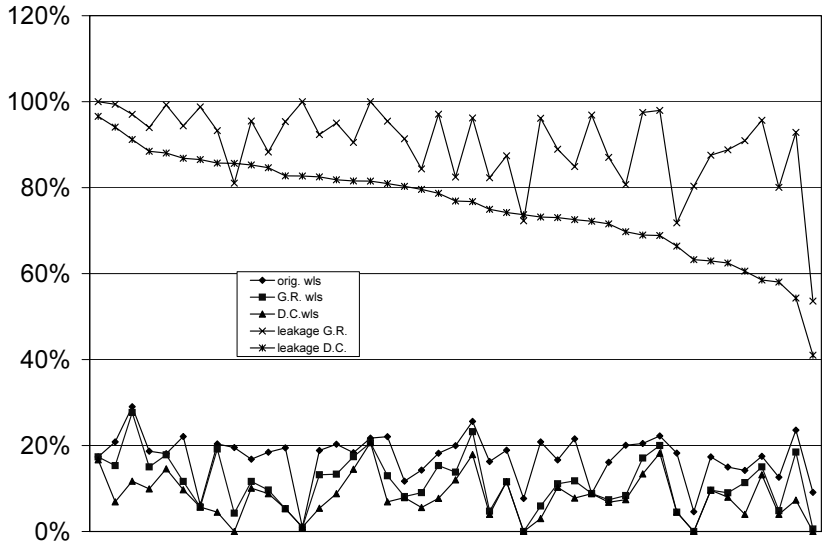


Figure 10: Leakage and WLS percentage on 43 large circuits with 22 PIs or more.

overhead and the area increase is only 2%.

The next seven columns show results by the divide-and-conquer approach where we set a 5% maximum delay constraint. In the genetic algorithm, we start with a population size of $N = 150$ and it converges after 50 generations. We are able to achieve, over the best MLV from 10,000 random vectors, 24% leakage saving with 7% area penalty on average. Although the average run time is 6X of the random search, we mention that this is mainly caused by the two circuits, *i8* and *des*. They have a couple of large tree circuits and therefore the frequently called dynamic programming takes considerably long time. Excluding these two circuits, the average run time for random search and the divide-and-conquer algorithm drop to 64.7s and 143s, respectively. More importantly, we see clearly the run time for random search increases exponentially to the number of primary input and linearly to the number of gates (columns 2,3,5). However, the run time for the divide-and-conquer approach grows at a much slower pace (column 12).

Finally, the last two columns compare our results with those reported in [1]. Because their detailed results are not available, we can only compare the average performance. In their experimental setup, the leakage reduction is compared with the average value among 1,000 random vectors. For a fair comparison, we also report in the last two columns the improvement of our approaches over the same baseline. Table 3 summarizes the performance

improvement in the control point insertion approach [1], our gate replacement algorithm, and the divide-and-conquer approach.

6 Conclusions

We study the MLV+ problem which 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 under the determined input vector when the circuit is at stand-by mode. The relaxation of circuit modification with changing its functionality enlarges the solution space of the IVC method. We show that MLV (and hence MLV+) problem is a hard problem and propose low-complexity heuristics to solve the MLV+ problem. The proposed algorithms are practical and effective in the sense that we do not need to change the design flow and re-do place-and-route. The experimental results show that this technique improves significantly the performance of IVC in leakage reduction at gate level with little area and delay overhead.

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] F. Aloul, A. Ramani, I. Markov and K. Sakallah, "PBS: A Backtrack-Search Pseudo-Boolean Solver and Optimizer", *Symposium on the Theory and Applications of Satisfiability Testing*, pp. 346-353, 2002.

[4] Mohab Anis, Mohamed Elmasry "Multi-Threshold CMOS Digital Circuits : Managing Leakage Power", *Springer*, October 2003.

[5] F. Assaderaghi, D. Sinitsky, S.A. Parke, J. Bokor, P.K. Ko, and C. Hu, "Dynamic Threshold-Voltage MOSFET(DTMOS) for ultra-low voltage VLSI", *IEEE Transaction on Electron Devices*, Vol. 44, pp. 414-422, 1997.

[6] S. Bobba and I.N. Hajj, "Maximum Leakage Power Estimation for CMOS Circuits", *IEEE Alessandro Volta Memorial Workshop on Low Power Design*, pp. 116, 1999.

[7] D. Blaauw, S. Martin, T.N. Mudge, Krisztia'n Flautner, "Leakage Current Reduction in VLSI Systems", *Journal of Circuits, Systems, and Computers* 11(6): 621-636 (2002)

[8] B.H. Calhoun, F.A. Honore, and A. Chandrakasan, “Design Methodology for Fine-Grained Leakage Control in MTCMOS”, *International Symposium on Low Power Electronics and Design*, pp. 104-109, 2003.

[9] Z. Chen, M. Johnson, L. Wei, and K. Roy, “Estimation of Standby Leakage Power in CMOS Circuits Considering Accurate Modeling of Transistor Stacks”, *International Symposium on Low Power Electronics and Design*, pp. 239-244, 1998.

[10] K. Chopra and S.B.K. Vrudhula, “Implicit Pseudo Boolean Enumeration Algorithms for Input Vector Control”, *ACM/IEEE Design Automation Conference*, pp. 767-772, 2004.

[11] 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.

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

[13] J. Halter, and F. Najm, “A Gate-Level Leakage Power Reduction Method for Ultra Low Power CMOS Circuits”, *IEEE Custom Integrated Circuits Conference*, pp 475-478, 1997.

[14] M.C. Johnson, D. Somasekhar, and K. Roy, “Models and Algorithms for Bounds on Leakage in CMOS Circuits”, *IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, pp. 714-725, 1999.

[15] J. Kao, S. Narendra, A. Chandrakasan, “Subthreshold Leakage Modeling and Reduction Techniques”, *Proceedings of ICCAD*, pp. 141-148, 2002

[16] D. Lee, W. Kwong, D. Blaauw, and D. Sylvester, “Analysis and Minimization Techniques for Total Leakage Considering Gate Oxide Leakage”, *ACM/IEEE Design Automation Conference*, pp. 175-180, June 2003.

[17] S. Mutoh, T. Douskei, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada, “1-V Power Supply High-Speed Digital Circuit Technology with Multi-threshold Voltage CMOS”, *IEEE Journal of Solid-State Circuits*, pp. 847-854, Aug. 1995.

[18] C. Neau and K. Roy, “Optimal Body Bias Selection for Leakage Improvement and Process Compensation over Different Technology Generations”, *International Symposium on Low Power Electronics and Design*, pp. 116-121, 2003.

[19] T. Kuroda, et al, “A 0.9V 150MHz 10mW 4mm2 2-D Discrete Cosine Transform Core Processor with Variable Threshold-Voltage(VT) Schemen”, *IEEE Journal of Solid-State Circuits*, pp. 1770-1779, Nov. 1996.

[20] 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”, *IEEE International Conference on Computer-Aided Design*, November 2003.

[21] H. Rahman and C. Chakrabarti, “A LEAKAGE ESTIMATION AND REDUCTION TECHNIQUE FOR SCALED CMOS LOGIC CIRCUITS CONSIDERING GATE-LEAKAGE”, *ISCAS*, 2004.

[22] L. Wei, Z. Chen, M. Johnson, K. Roy, and V. De, “Design and Optimization of Low Voltage High Performance Dual Threshold CMOS Circuits”, *Proceedings of DAC*, pp. 489-494, 98.

[23] Y. Ye, S. Borker, and V. De, “A New Technique for Standby Leakage Reduction in High-Performance Circuits”, *Symposium on VLSI Circuits*, pp. 40-41.9, 1998.

[24] M.R. Garey and D.S. Johnson, “Computers and Intractability, A Guide to the Theory of NP-Completeness”, *Freeman Company*, 2001.

[25] G.D. Hachtel and F. Somenzi, “Logic Synthesis and Verification Algorithms”, *Kluwer Academic Publishers*, 1996.

[26] Saeyang Yang, “Synthesis and Optimization Benchmarks User Guide”, 2002, ftp://mcnc.mcnc.org.

[27] E. Sentovich, et al., “SIS: A System for Sequential Circuit Synthesis,” *Electronics Research Laboratory Memorandum, U.C.Berkeley*, No. UCB/ERL M92/41.

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

circuit	pi	gate	exhaustive		gate replace			divide-and-conquer				
	#	#	leak(nA)	wls	imprv	wls	ar inc	imprv	wls	# tr	# cg	ar inc
b1	3	13	2195	23%	2%	15%	5%	2%	10%	5	0%	5%
cm42a	4	25	2941	0%	0%	0%	0%	8%	0%	18	4%	8%
C17	5	6	831	17%	43%	0%	17%	43%	0%	4	0%	17%
cm82a	5	28	5017	21%	29%	4%	12%	40%	1%	10	4%	18%
decod	5	22	1921	0%	0%	0%	0%	8%	0%	21	5%	3%
cm138a	6	19	1760	0%	0%	0%	0%	1%	0%	12	5%	5%
z4ml	7	66	12246	24%	25%	11%	11%	37%	4%	20	5%	17%
f51m	8	136	26038	26%	37%	7%	12%	48%	4%	25	3%	14%
9symml	9	166	34018	26%	20%	17%	5%	38%	8%	18	8%	14%
alu2	10	356	64153	21%	2%	20%	0%	21%	5%	89	7%	11%
x2	10	44	6159	9%	15%	2%	3%	12%	2%	18	9%	10%
cm85a	11	38	4925	8%	14%	3%	3%	13%	3%	16	0%	3%
cm151a	12	34	5745	24%	9%	18%	4%	3%	18%	5	3%	5%
alu4	14	728	133127	25%	1%	21%	1%	15%	4%	166	7%	10%
cm162a	14	45	6947	18%	2%	9%	3%	0%	9%	13	4%	12%
cu	14	49	6182	12%	16%	6%	2%	9%	5%	21	6%	7%
cm163a	16	43	6376	19%	2%	9%	3%	1%	9%	11	5%	13%
cmb	16	42	5405	10%	11%	5%	2%	4%	4%	8	2%	6%
parity	16	75	12764	20%	11%	15%	5%	15%	7%	15	7%	20%
pm1	16	39	3474	3%	0%	0%	1%	-2%	0%	16	3%	3%
t481	16	1945	251184	2%	1%	1%	0%	26%	0%	17	2%	1%
tcon	17	41	6491	20%	43%	0%	14%	41%	0%	9	2%	17%
pcle	19	74	12594	20%	32%	4%	6%	32%	4%	22	0%	6%
sct	19	92	11811	18%	14%	9%	4%	10%	6%	24	4%	6%
cc	21	48	5823	13%	6%	10%	1%	6%	9%	22	0%	1%
cm150a	21	72	12270	15%	4%	14%	1%	1%	10%	9	7%	10%
Average				15%	13%	8%	4%	17%	4%		4%	9%

Table 2: Results on 43 large circuits with primary inputs more than 22.

circuit	pi	gate	random search (10k)			gate replacement (G.R.)				divide-and-conquer (D.C.)							over 1K aver	
	#	#	leak(nA)	time(s)	wls(%)	imprv(%)	time(s)	wls(%)	area(%)	imprv(%)	time(s)	wls(%)	#tree	#gates/tree	# cg	area(%)	G.R.(%)	D.C
cordic	23	102	18434.0	9.9	21.6	15.1	0.01	11.8	5.7	27.4	10.1	7.8%	52	3.1	7%	9.3	28.4	3
ttt2	24	207	33801.5	22.7	18.4	9.5	0.02	17.4	4.4	18.4	72.6	14.5%	43	4.7	6%	9.6	30.9	3
i1	25	39	5250.6	5.4	7.7	27.7	0	0.0	4.3	26.3	6.0	0.0%	16	2.1	3%	5.1	45.5	4
pcler8	27	90	14670.1	10.0	16.7	11.1	0.01	11.1	4.0	27.0	14.9	10.3%	31	2.4	0%	4.0	35.2	4
c8	28	164	26083.0	17.4	19.5	19.0	0.01	4.3	8.4	14.4	21.5	0.0%	38	5.9	8%	6.9	31.7	2
C6288	32	2400	480084.2	222.0	29.0	2.9	0.11	27.7	1.9	8.8	398.7	11.7%	1424	1.7	29%	27.3	7.0	1
comp	32	163	28322.3	15.2	22.1	5.6	0.01	11.7	2.4	13.2	85.4	9.7%	77	3.4	2%	5.4	34.1	3
C1908	33	615	117029.6	57.2	20.5	2.5	0.02	17.1	0.9	31.0	66.0	13.4%	218	2.9	10%	10.1	6.4	3
my_adder	33	225	40842.1	21.0	22.2	2.0	0.02	20.0	1.5	31.1	32.1	18.2%	95	2.8	7%	6.4	8.9	3
term1	34	363	60460.5	37.3	18.5	11.7	0.02	9.6	4.0	15.4	160.0	8.8%	75	6.8	5%	8.8	23.9	2
count	35	144	22445.4	15.2	17.4	0.0	0.01	17.4	0.0	3.4	14.2	16.7%	37	4.2	2%	2.4	0.0	1
C432	36	200	38101.4	20.1	15.0	11.2	0.01	9.0	3.3	37.5	24.7	8.0%	79	4.1	6%	8.9	21.6	4
unreg	36	113	18188.4	12.7	19.5	4.6	0.01	5.3	3.1	17.3	84.4	5.3%	18	6.3	2%	4.9	20.1	3
too_large	38	582	107888.1	61.4	17.4	12.5	0.05	9.6	2.2	37.1	80.1	9.6%	113	5.2	7%	10.9	24.5	4
b9	41	111	16100.3	12.8	11.7	8.6	0.01	8.1	2.0	19.7	68.0	7.9%	34	3.3	4%	8.7	30.1	3
C1355	41	517	91739.0	50.7	22.1	4.5	0.02	13.0	1.4	19.1	95.0	6.9%	265	2.0	15%	13.1	12.1	2
C499	41	532	95292.0	48.3	20.3	5.0	0.05	13.3	2.2	18.2	84.5	8.8%	197	2.7	7%	5.8	16.8	2
cht	47	232	38560.8	25.3	16.8	4.5	0.02	11.6	3.7	14.7	22.8	10.1%	66	3.5	2%	3.3	18.4	2
apex7	49	239	41955.1	26.0	20.1	19.3	0.02	8.4	5.8	30.3	25.6	7.4%	82	2.9	3%	11.1	26.9	3
C3540	50	1136	218977.1	115.0	18.2	2.9	0.08	15.3	1.3	21.3	133.8	7.7%	381	3.0	15%	2.1	11.5	2
x1	51	295	45351.2	32.8	16.3	17.7	0.02	4.7	4.8	25.0	105.9	4.0%	61	4.8	7%	11.9	32.1	3
C880	60	354	61978.8	35.8	18.9	12.6	0.04	11.6	4.1	25.8	39.9	11.6%	115	3.1	13%	10.8	21.7	3
dalu	75	1865	349299.8	187.5	25.6	3.8	0.15	23.2	1.4	23.2	194.9	17.9%	321	5.8	8%	14.2	29.1	4
example2	85	286	51036.6	32.6	17.5	4.3	0.02	15.0	1.4	41.5	28.9	13.2%	110	2.6	2%	9.8	11.3	4
i9	88	510	88469.6	63.9	1.0	0.0	0.04	1.0	0.0	17.3	156.0	1.0%	113	4.5	3%	2.1	0.0	5
x4	94	378	61336.3	46.4	18.3	28.2	0.03	4.5	5.3	33.6	206.5	4.5%	110	3.4	11%	8.6	40.1	4
i3	132	92	16166.9	14.9	21.7	0.0	0.00	21.7	0.0	18.5	0.0	20.7%	6	15.3	0%	0.0	0.0	2
i5	133	269	44848.1	34.3	12.6	19.9	0.02	4.8	2.9	42.0	45.6	4.0%	68	4.0	2%	7.8	35.8	5
i8	133	1898	305924.5	224.4	14.2	9.1	0.15	11.4	0.8	39.4	7591.3	4.0%	259	7.3	6%	6.3	43.5	6
apex6	135	710	126523.6	86.1	20.8	3.9	0.06	5.9	2.1	26.8	399.5	3.0%	215	3.3	10%	5.7	11.4	3
rot	135	601	109944.1	67.1	20.0	17.5	0.06	13.8	5.5	23.1	403.3	12.0%	208	2.9	10%	12.7	23.5	2
x3	135	742	116641.0	89.5	14.3	15.6	0.07	9.0	3.2	20.4	384.4	5.6%	192	3.9	8%	10.0	29.7	3
i6	138	340	47021.1	47.3	9.1	46.4	0.03	0.6	2.1	59.0	89.8	0.0%	71	4.8	1%	3.0	68.9	7
frg2	143	1030	165090.4	136.0	16.1	12.9	0.11	7.4	3.2	28.4	176.5	6.8%	244	4.2	5%	7.4	28.0	4
pair	173	1538	270729.8	160.9	18.9	7.6	0.14	13.2	2.4	17.5	366.0	5.4%	434	3.5	12%	12.0	14.9	2
C5315	178	1777	343295.9	188.3	18.7	6.0	0.15	15.0	2.0	11.5	534.5	9.9%	532	3.3	12%	15.1	11.6	1
i4	192	136	22699.8	22.8	8.8	3.1	0.01	8.8	0.4	27.8	34.6	8.8%	6	22.7	0%	4.6	28.3	4
i7	199	405	58431.5	58.4	6.2	1.2	0.04	5.7	0.2	13.5	117.9	5.7%	76	5.3	2%	1.1	37.7	4
i2	201	109	13174.8	22.1	4.6	19.7	0.01	0.0	2.2	36.8	36.1	0.0%	12	9.1	4%	3.6	36.1	4
C7552	207	2801	515320.2	293.3	20.8	0.6	0.18	15.3	0.3	5.9	726.0	6.9%	908	3.1	15%	16.1	20.6	2
C2670	233	807	155992.3	94.5	18.1	0.8	0.09	17.8	0.2	11.9	98.6	14.6%	235	3.4	11%	9.9	5.4	1
des	256	3995	931447.4	471.2	23.6	7.2	0.24	18.5	2.5	45.7	8502.6	7.3%	847	4.7	11%	14.2	17.6	5
i10	257	2281	440552.2	261.6	20.4	6.7	0.2	19.2	1.9	14.3	162.8	4.5%	695	3.3	14%	6.1	11.7	1
Average				80.9	17%	10%	0.05	11%	2%	24%	510.2	9%			6%	7%	23%	3

Table 3: Average performance comparison with [1] algorithm.

	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%