A Fast Probability-Based Algorithm for Leakage Current Reduction Considering Controller Cost*

Tsung-Yi Wu Jr-Luen Tzeng Kuang-Yao Chen

Department of Electronic Engineering
National Changhua University of Education
Changhua, Taiwan
tywu@cc.ncue.edu.tw, m100@ms5.hinet.net, muscle9839@gmail.com

Abstract— Because the leakage current of a digital circuit depends on the states of its logic gates, assigning a *minimum leakage vector* (MLV) to the primary inputs and the flip-flops' output pins of the circuit that operates in the sleep mode is a feasible technique for leakage current reduction. In this paper, we propose a novel probability-based algorithm and technique that can rapidly find an MLV. Unlike most traditional techniques that ignore the leakage current overhead of the newborn MLV controller, our technique can take this overhead into account. Ignoring this overhead during solution exploration may bring a side effect that is misrecognizing a non-optimum solution as an optimum one. Experimental results show that our algorithm can reduce the leakage current up to 48% and can find the optimum solutions on 22 out of 26 small MCNC benchmark circuits.

I. INTRODUCTION

High leakage current is one of the special phenomenons of ultra deep sub-micron (UDSM) chips. Therefore, if the leakage current of a UDSM chip used in a portable electronic device is not properly controlled, the chip will shorten the operating time of the device's battery.

Designing power-gating CMOS logic gates for low leakage current chips is proposed in [1]–[4], etc. The new architecture of a power-gating logic gate has additional control pins that are used to enable or disable the power supply of the gate. During the sleep mode, leakage current of the gate is very small because the MOS transistors connecting to VDD and ground are disabled. The penalties of power-gating techniques are area and wake-up time overheads, etc.

The multi-threshold CMOS (MTCMOS) technique [4]–[9] provides a solution to carry out high performance and low leakage power requirements of chips. A low V_t (threshold voltage) gate has high leakage power and short delay while a high V_t gate has lower leakage power and longer delay. In order to achieve high performance and low leakage power requirements, we can apply low V_t gates to critical paths and put high V_t gates in non-critical paths as many as possible.

The leakage current of a gate depends on the logic states of its input pins. TABLE I shows one such example. In this example, the leakage currents of a 65nm 2-input NAND gate and a 65nm 2-input NOR gate are reported by HSPICE simulator. The BSIM4 model card used in the simulator is *Berkeley Predictive Technology Model* (BPTM) [10]. If the input vector is (0, 0), NAND gate has the minimum leakage current while NOR gate has the maximum leakage current. Moreover, the leakage currents corresponding to input vector (0, 1) and to input vector (1, 0) are different. Therefore, state assignments of primary inputs and pin reordering of logic gates are feasible techniques for reducing the leakage current of a circuit that operates in the sleep mode [11]–[19].

For a digital circuit operating in the sleep mode, finding a vector (i.e. minimum leakage vector (MLV)) of the primary inputs that can cause the circuit to consume the minimum leakage current is an NP-hard problem [20]. In general, the problem is called MLV problem. For convenience of processing, F. Aloul et al. [11] and A. Abdollahi et al. [12] formulated MLV problems as satisfiability (SAT) problems. F. Gao et al. [13] formulated MLV problems as integer linear programming (ILP) problems. Because an ILP problem is also an NP-hard problem, they furthermore proposed a heuristic mixed-integer linear programming (MLP) method to fast solve MLV problems. L. Yuan et al. [14] used a gate replacement technique and a divide-and-conquer approach to reduce the total leakage current of a design that operates in the sleep mode. The algorithm used by L. Yuan has low time complexity. J. P. Halter et al. [15] developed an efficient algorithm that determines an MLV using a sampling of random vectors. D. Lee et al. [16][17] and K. Chopra et al. [19] used a branch-and-bound approach and implicit pseudo Boolean enumeration algorithms, respectively, to find a better input vector for reducing the leakage current of a circuit. For solving MLV problem, R. M. Rao et al. [18] proposed a greedy search that is based on the controllability of nodes in a circuit and uses the functional dependencies among cells in the circuit to guide the search.

In this paper, we propose a probability-based algorithm to reduce the leakage current of a digital circuit that operates in the sleep mode. By using the expected value of each explored solution's leakage current as a cost function, our algorithm can rapidly construct an MLV controller for a circuit. The new circuit that consists of the original circuit and the MLV controller consumes less leakage current than the original circuit and nearly does not consume any dynamic power

TABLE I
LEAKAGE CURRENT OF LOGIC GATES

Gate Type	Input Vector			
	(0, 0)	(0, 1)	(1, 0)	(1, 1)
NAND	0.79 nA	5.38 nA	3.03 nA	7.23 nA
NOR	7.71 nA	2.38 nA	4.78 nA	0.72 nA

* This research is supported by the National Science Council of R.O.C. under contract no. NSC 95-2221-E-018-024.

1-4244-0630-7/07/$20.00 ©2007 IEEE.

under the sleep mode. Experimental results show that our algorithm can find the optimum solutions on 22 out of 26 small MCNC benchmark circuits.

In the next section, we describe our algorithm. Some experimental results on a set of benchmarks are presented in Section III. Finally, we conclude this paper and describe future work in Section IV.

II. Probability-based Algorithm

Before we disclose the leakage current reduction algorithm, we will introduce notations, terminologies, definitions and a theorem that are used in the algorithm.

A. Notations, Terminologies, Definitions and a Theorem

We are given a logic design D with m gates $G = \{g_1, g_2, \ldots, g_m\}$, where g_h is the *h-th* gate of D. *Let* $CPV = (i_1, i_2, \ldots, i_p, fo_1, fo_2, \ldots, fo_q)$ be a control-point vector (CPV) for D in the sleep mode, where i_h is the logic value of the *h-th* primary input of D, and fo_h is the logic value of the output pin of the *h-th* flip-flop. Therefore, i_h and fo_h are **always ZERO**, **always ONE**, or **unfixed**. For example, the primary input named RESET is always ZERO[†] and the output of a data register may be unfixed. An unfixed control-point may be logic 0 in the current sleep mode while it may be logic 1 in the next sleep mode. In this paper, an unfixed value is denoted as *X*. Always ZERO and always ONE are denoted as 0 and 1, respectively.

A given control-point vector *CPV* is *nondeterministic* if it has at least one element that is *X*; otherwise *CPV* is *deterministic*. A deterministic control-point vector can be denoted as *DCPV*. The set of all *feasible deterministic control-point vectors* for a given *CPV* is denoted as $FDCPV(CPV) = \{DCPV_1, DCPV_2, \ldots, DCPV_k\}$, where each member of *FDCPV(CPV)* is deterministic. For example, if $CPV = (1, X, 0, X)$, then $FDCPV(CPV) = \{DCPV_1, DCPV_2, DCPV_3, DCPV_4\} = \{(1, 0, 0, 0), (1, 0, 0, 1), (1, 1, 0, 0), (1, 1, 0, 1)\}$.

The logic states of all input pins and all output pins of each gate in a design are deterministic if the control-point vector of the design is deterministic. Therefore, the exact leakage current of a design associated with any *DCPV* can be computed.

Definition 1 *Lkg_of_dsgn(D, CPV)* is defined as the leakage current of D with the control-point vector *CPV*. Obviously, *Lkg_of_dsgn(D, CPV)* is unknown if *CPV* is nondeterministic. However, the expected value of the leakage current for a nondeterministic control-point vector is not unknown because it can be calculated.

Definition 2 Let *E[Lkg_of_dsgn(D, CPV)]* be the expected value of *Lkg_of_dsgn*(*D, CPV*).

Definition 3 *Lkg_of_gate*(g_h, *D, CPV)* is defined as the leakage current of the gate g_h *of D* with the control-point vector *CPV*. Let *E[Lkg_of_gate*(g_h, *D, CPV)]* be the expected value of *Lkg_of_gate*(g_h, *D, CPV)*.

[†] In a chip, there are many always ZERO and always ONE signals, for example, JTAG signals, control signals for memory BIST circuits, etc.

Theorem 1 Given any *CPV*, then $E[Lkg\_of\_dsgn(D, CPV)] = \sum_{j=1}^{m} E[Lkg\_of\_gate(g_j, D, CPV)]$

Proof: Assume $FDCPV(CPV) = \{DCPV_1, DCPV_2, \ldots, DCPV_k\}$, and let $P(V=DCPV_h)$ be the probability that $DCPV_h$ is chosen from *FDCPV(CPV)*, where V is a random variable. By probability theorem, we know that

$$E[Lkg\_of\_dsgn(D, CPV)] = \sum_{i=1}^{k} [P(V=DCPV_i) * Lkg\_of\_dsgn(D, DCPV_i)] \quad (1)$$

$$= \sum_{i=1}^{k} \{P(V=DCPV_i) * \sum_{j=1}^{m} [Lkg\_of\_gate(g_j, D, DCPV_i)]\}$$

$$= \sum_{j=1}^{m} \sum_{i=1}^{k} [P(V=DCPV_i) * Lkg\_of\_gate(g_j, D, DCPV_i)]$$

$$= \sum_{j=1}^{m} E[Lkg\_of\_gate(g_j, D, CPV)]$$

B. Time Complexity of Computing $E^+[Lkg\_of\_dsgn(D, CPV)]$

In a design D, the probabilities of logic values of all input pins of all gates are computable if the probabilities of logic ONE and logic ZERO of all primary inputs and all flip-flops' outputs are known. *The worst-case time complexity of this computation is only O(m) if D has m gates and each gate of D only has probability-independent input pins.* The reason is that the probability of being ONE or being ZERO of the output pin of each gate can be computed by $O(2^p)$, where p is the number of the input pins of the gate. $O(2^p)$ can be rewritten as O(1) because p is a small constant for each gate in a typical standard cell library. Therefore, calculating the probability data of m gates takes O(m) (= $O(2^p)$*O(m)). In our algorithm, all input pins of each gate are treated as probability-independent pins for achieving the requirement of O(m) time complexity. Hence, in our algorithm, the computed probability data of those gates that have probability-dependent pins are approximate values rather than exact ones.

Fig. 1 illustrates how to calculate the probability data of net C. In this figure, let P(C=0) (P(C=1)) be the probability of net C being logic ZERO (ONE). The probability data of each primary input (i.e. TEST, A, and B) are given by us. We can directly calculate the probability data of net C by taking O(1) (= $O(2^2)$) because the probability data of ports A and B are known. As with the probability data of net C, then, we are able to calculate the probability data of ports Y and Z. Fig. 2 shows the final result.

Similarly, we can calculate the expected value of leakage current of a gate by constant time (O(1)) if the probabilities of logic values of all input pins of the gate are known. By Theorem 1, we know that the expected leakage current of a

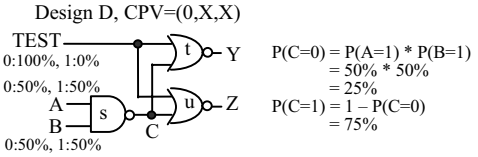

Fig. 1: An illustration of how to calculate the probability data of net C.

design is the sum of the expected values of all *m* gates' leakage currents, so the total time complexity of computing *E[Lkg_of_dsgn(D, CPV)]* is *O(m)* (= O(1)*O(m)+O(m), where the last O(m) is consumed in computing the probabilities of logic values of all input pins of all gates). However, if we use (1) rather than Theorem 1 to calculate the expected leakage current of this design, the time complexity of the worst case is $O(m*2^n)$ rather than *O(m)*, where *n* is the dimension of *CPV* of the design. The reason is that the value of *k* in (1) is 2^n under the worst case.

E[Lkg_of_dsgn(D, CPV)] is a cost function used in our algorithm to judge the quality of an explored *CPV*. Fig 2 illustrates how to compute the value of *E[Lkg_of_dsgn(D, CPV)]*. In this illustration, the probability data have been computed and labelled in each net. The leakage current data of NAND gate and NOR gate are listed in Table I. Because *E[Lkg_of_dsgn(D, CPV)]* used in our algorithm may be an approximate value rather than an exact one if design *D* has any gate that has probability-dependent input pins, we use $E^+[Lkg\_of\_dsgn(D, CPV)]$ notation instead of *E[Lkg_of_dsgn(D, CPV)]* notation in the algorithm in order to avoid confusion of readers.

C. Algorithm

In this subsection, we propose an algorithm for leakage current reduction. For an inputted original design D_O with a given CPV_I (i.e. Inherent CPV) in the sleep mode, ALGORITHM 1 tries to find a *deterministic* CPV_M (i.e. Minimum Leakage CPV) that is used to construct a new design D_N from D_O such that $E[Lkg\_of\_dsgn(D_N, CPV_I)]$ is as less as possible. Under the normal mode, D_N and D_O are functionally equivalent.

CPV_I is given by circuit designers. If the designers know that some primary input signal is always ZERO (always ONE) then its corresponding element in CPV_I is assigned 0 (1) by them. For a primary input that is neither always ZERO nor always ONE, the designers should assign *X* to its corresponding element of CPV_I.

In ALGORITHM 1, the initial value of CPV_M is set to be (*X*, *X*,…, *X*) (line 1) and the initial value of *the probability change rate*, *Rt*, is set as 10% (line 2). For each *X* in CPV_M, its initial probabilities of being ZERO and being ONE are both 50% (i.e. 0.5). In the *For Loop* block, ALGORITHM 1 sequentially scans and processes each element e_k of CPV_M. During each processing procedure, the algorithm employs a leakage cost function (i.e. $E^+[Lkg\_of\_dsgn(D_O, CPV_M)]+Lkg(CPV_I, CPV_M)$[‡] shown in lines 8 and 11) as the criterion to decide whether to decrease (line 6) or increase

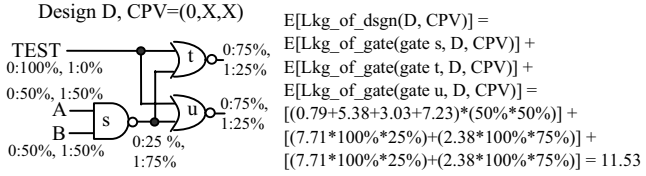

Fig. 2: An illustration of how to calculate E[Lkg_of_dsgn(D, CPV)].

(line 10) the e_k's probability of being ONE by *Rt*. After many times of the *While Loop* iterations, a deterministic CPV_M is obtained because there is no *X* in CPV_M, and then ALGORITHM 1 uses CPV_I, CPV_M and D_O to construct D_N (line 17). D_N consists of D_O and a CPV controller. Finally, our algorithm applies pin-reordering technique to D_N for realizing the further optimization.

There is a function, *INC_PROB,* used in the algorithm. The *INC_PROB* function-call for the element e_k of CPV_M[§] (i.e. $INC\_PROB(CPV_M, e_k, ZERO, Rt)$) that appears in line 6 of ALGORITHM 1 increases e_k*'s* probability of being ZERO by *Rt*. After calling the *INC_PROB* function, an updated CPV_M will be returned. Note that the updated probability cannot be greater than 100%. The value of e_k is changed from *X* to *ZERO* (*ONE*) if the probability of e_k being *ZERO* (*ONE*) arrives at 100%. Similarly, the value of e_k is changed from *ZERO* (*ONE*) to *X* if the probability of e_k being *ZERO* (*ONE*) isn't 100% anymore.

ALGORITHM 1

Input: $CPV_I = (i_{i1}, i_{i2},\ldots, i_{ip}, fo_{i1}, fo_{i2},\ldots, fo_{iq})$: inherent CPV;
D_O : the original design;
Output: $CPV_M = (i_{m1}, i_{m2},\ldots, i_{mp}, fo_{m1}, fo_{m2},\ldots, fo_{mq})$: min. leakage CPV;
D_N : the new design;

1. CPV_M = (X, X,…, X);
2. Rt = 10%;
3. While (CPV_M has any element that value is X)
4. For each element e_k of CPV_M
5. $Org\_CPV_M = CPV_M$; /* Keep the original CPV_M for restoration. */
6. $CPV_M = INC\_PROB(CPV_M, e_k, ZERO, Rt)$;
7. $Next\_CPV_M = CPV_M$;
8. $Lkg_{min} = E^+[Lkg\_of\_dsgn(D_O, CPV_M)] + Lkg(CPV_I, CPV_M)$;
9. $CPV_M = Org\_CPV_M$; /* Restore the original CPV_M for next test. */
10. $CPV_M = INC\_PROB(CPV_M, e_k, ONE, Rt)$;
11. If ($Lkg_{min} > E^+[Lkg\_of\_dsgn(D_O, CPV_M)] + Lkg(CPV_I, CPV_M)$)
12. Then $Next\_CPV_M = CPV_M$;
13. $CPV_M = Next\_CPV_M$; /* Update CPV_M. */
14. End For Loop
15. Rt = Rt + 10%;
16. End While Loop
17. Use CPV_I, CPV_M and D_O to construct D_N; /* Controller synthesis */
18. Reconstruct D_N by using pin-reordering technique;

The e_k's probability of being ZERO or ONE may be 0%, 10%, 20%,…, 90%, or 100%. *Line 15 of ALGORITHM 1 controls the convergence rate of this algorithm.* Under the worst case, ALGORITHM 1 executes 10 times (= O(1)) of *While Loop* iterations. By the discussion of the previous sub-section, we know that judging a CPV_M (lines 8 and 11) has *O(m)* time complexity. The worst-case time complexity of ALGORITHM 1 is *O(n*m)* (= O(n)*O(m)*O(1)) because *For Loop* iterates *n* times during each *While Loop* iteration, and each iteration of *For Loop* takes *O(m)*. In fact, an iteration of *For Loop* takes much less than *O(m)* for a general design D_O because only a small part of the gates in D_O needs to update the expected values of their leakage currents.

[‡] The function $Lkg(CPV_I, CPV_M)$ returns the leakage current consumed by the vector translation circuit (i.e. CPV controller) that is used to translate CPV_I into CPV_M.

[§] In our algorithm, CPV_M is not a traditional mathematical vector. It has special data structure. If the value of e_k of CPV_M is *X*, then e_k has an additional record that describes its probabilities of being 0 and being 1.

Fig. 3 illustrates the solution exploration of ALGORITHM 1. In this illustration, we assume that D_O is the design shown in Fig. 2. At the first branch of the search tree, ALGORITHM 1 decides to increase *P(B=0)* to be 60% (from 50%) because this decision has less value of $E^+[Lkg\_of\_dsgn(D_O, CPV_M)]+Lkg(CPV_I, CPV_M)$ than the decision of increasing *P(B=1)* to be 60%. Similarly, ALGORITHM 1 decides to increase *P(A=0)* to be 60% in the next branch node. After many times of explorations, a deterministic CPV_M will be constructed.

D. Controller for Control-Point Vector

In this subsection, we will introduce how to use sleep-gating technique to implement CPV controllers.

If we feed ALGORITHM 1 with CPV_I and D_O, then CPV_M will be constructed by the algorithm. It is necessary to convert CPV_I into CPV_M by an extra circuit (i.e. CPV controller) if $CPV_I \neq CPV_M$. Fig. 4 presents an illustration of how to construct a CPV controller. Fig. 4(a) and Fig. 4(b) show the original design (D_O) and the new one (D_N), respectively. Because the value (i.e. *X*) of input *A* in CPV_I is different from the value (i.e. 0) of input *A* in CPV_M, the AND gate *g* is inserted into D_O for constructing D_N.

By observing Fig. 4, it is clear that D_O and D_N have the same functionality in the normal mode because signal *Sleep* is in logic ZERO. In the sleep mode, due to signal *Sleep* being logic ONE, the functions of D_O and D_N are the same if input *A* is logic ZERO. So the circuit shown in Fig. 4(b) achieves ZERO state assignment on the primary input *A*.

E. Our Technique vs. Traditional Techniques

Unlike many other papers that ignore the overhead of the leakage current of a CPV controller, ALGORITHM 1 can count the overhead of the controller during solution space exploration. Ignoring the overhead may bring a side effect that is misrecognizing a non-optimum solution as an optimum one. We will give an example in the next paragraph.

Because ALGORITHM 1 can allow circuit designers to input a CPV_I but traditional techniques cannot, our technique usually can find a better solution than the traditional ones. We use a very simple design (i.e. a NAND gate) shown in Fig. 5 to explain this fact. The traditional techniques will find a CPV_M = (0, 0) for this design because NAND gate has the minimum leakage current when *A=0* and *B=0*. However, the CPV_M found by our algorithm is (1, 0) (i.e. CPV_M = (1, 0)) rather than (0, 0) because our algorithm has the information that CPV_I is (1, 0). If designers want to implement the simple design with CPV_M = (0, 0), they must insert an AND gate

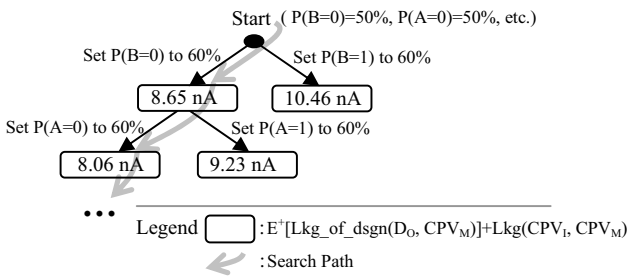

Fig. 3: Solution exploration for the design shown in Fig. 2.

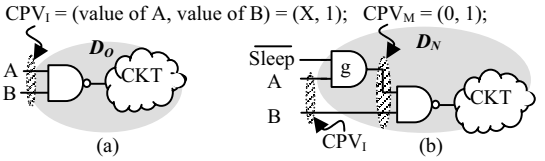

Fig. 4: (a) Design D_O. (b) Design D_N.

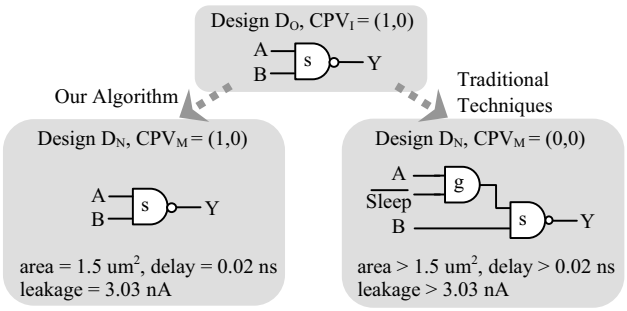

Fig. 5: An example to illustrate the difference between our algorithm and the traditional techniques.

named *g* to the original design as shown in Fig. 5. Comparing the D_N*s* constructed by our technique and by the traditional techniques, we find that our D_N has less area, delay, and leakage current than their D_N.

III. EXPERIMENTS

We have implemented a leakage current reduction system named *Leakage Current Solver* on an IBM xSeries 235 server that has two 3.06GHz Intel Xeon processors. Leakage Current Solver employs ALGORITHM 1 to construct a new design that consumes less leakage current than the original one. *The new design nearly does not consume any dynamic power under sleep mode* because the switching signals on the primary inputs cannot pass through the CPV controller to trigger the internal gates of the design.

In order to evaluate the quality of our algorithm and system, some representative MCNC and ISCAS benchmark circuits are used in the experiments.

A 90nm CMOS standard cell library is used in the experiments. It is constructed by shrinking TSMC 0.18um standard cells. BPTM [10] process model is used in the leakage library characterization program of Leakage Current Solver. All gate level netlists of the benchmarks used in the experiments are synthesized by Design Compiler® of Synopsys, Inc.

In the first experiment, we compare the experimental results of an Exhaustive Search Program (ESP) with those of our algorithm. The results are shown in TABLE II. In this experiment, the functions of pin-reordering (line 18 of ALGORITHM 1) and controller synthesis (line 17 of ALGORITHM 1) of ALGORITHM 1 are turned off. In other word, ALGORITHM 1 is treated as an MLV generator. All benchmark circuits used in this experiment are small MCNC benchmark circuits rather than large ones because ESP cannot solve a large circuit within reasonable CPU time. These benchmark circuits are also used in the experiment of [14]. Fortunately,

TABLE II
EXPERIMENTAL RESULTS FOR EXHAUSTIVE SEARCH AND ALGORITHM 1
(† Functions of pin reordering and controller synthesis are turned off.)

Bench-marks	#PI	Area (um^2)	Exhaustive Search			ALGORITHM 1†		Comparison		
			leakage (nA)		CPU time (s)	leakage (nA) (L3)	CPU time (s)	CPU time red.	L3–L1 / L1	L2–L3 / L2
			min. (L1)	max. (L2)						
b1	3	27	2.6	3.6	0.00	2.6	0.00	–	0%	28%
cm42a	4	62	4.7	6.1	0.00	4.7	0.00	–	0%	23%
C17	5	17	1.5	2.3	0.00	1.5	0.00	–	0%	35%
cm82a	5	55	5.6	6.8	0.00	5.6	0.00	–	0%	18%
decod	5	78	4.7	7.0	0.00	4.7	0.00	–	0%	33%
cm138a	6	55	3.4	6.5	0.01	3.4	0.01	0%	0%	48%
z4ml	7	59	6.0	7.1	0.01	6.0	0.01	0%	0%	15%
f51m	8	274	27.3	31.9	0.05	28.0	0.02	60%	2.6%	12%
9symml	9	140	14.7	16.7	0.06	14.9	0.01	83%	1.4%	11%
alu2	10	885	89.9	98.6	0.72	90.0	0.11	84%	0.1%	9%
x2	10	113	8.7	13.7	0.08	8.7	0.01	88%	0%	36%
cm85a	11	121	11.1	16.2	0.18	11.1	0.01	94%	0%	31%
cm151a	12	65	6.4	7.7	0.19	6.4	0.01	95%	0%	17%
alu4	14	2274	222.1	246.3	90.79	222.1	0.44	100%	0%	10%
cm162a	14	111	9.7	13.7	1.31	9.7	0.02	98%	0%	29%
cu	14	126	9.5	15.0	1.40	10.2	0.03	98%	7.4%	32%
cm163a	16	98	8.2	12.2	4.76	8.2	0.01	100%	0%	33%
cmb	16	80	4.7	8.8	3.52	4.7	0.01	100%	0%	47%
parity	16	159	16.4	20.1	9.70	16.4	0.03	100%	0%	18%
pm1	16	100	7.0	12.9	4.55	7.0	0.03	99%	0%	46%
t481	16	88	8.1	11.9	4.08	8.1	0.02	100%	0%	32%
tcon	17	65	6.4	7.7	0.12	6.4	0.01	92%	0%	17%
pcle	19	165	14.8	19.4	66.46	14.8	0.03	100%	0%	24%
sct	19	179	15.4	22.6	69.78	15.4	0.04	100%	0%	32%
cc	21	126	10.4	16.8	192.00	10.4	0.02	100%	0%	38%
cm150a	21	115	11.6	13.7	334.60	11.6	0.03	100%	0%	15%
Average	**12**	**216**	**20.4**	**24.8**	**30.17**	**20.5**	**0.04**	**85%**	**0.4%**	**27%**

our algorithm can find the optimum solutions on all benchmark circuits except for *f51m*, *9symml*, *alu2* and *cu* (column 10). If the function of pin-reordering in ALGORITHM 1 is turned on, the total leakage current of all circuits constructed by our algorithm is less than the total ESP's minimum and maximum leakage currents by 7% and 24%, respectively. This result is shown in TABLE III.

In the second experiment, a Random Search Program (RSP) [13][14][15][16][18] was implemented as a basis of comparison to our algorithm for 12 large benchmark circuits. For each benchmark circuit, RSP randomly generates large number of control-point vectors and records the leakage current data associated with these control-point vectors.

TABLE IV summarizes the result of the second experiment. Column 2 shows the number of primary inputs and the number of flip-flops in each circuit. Columns 4 and 5 show the critical path (CP) delay and the number of randomly generated control-point vectors, respectively. The minimum and maximum leakage currents obtained by RSP are listed in column 6 and column 7, respectively. Column 8 shows the CPU time consumed by RSP. Column 9 shows the leakage current of the original circuit (without CPV controller) that is

TABLE III
COMPARISON BETWEEN EXPERIMENTS OF ESP AND ALGORITHM 1

Exhaustive Search		ALGORITHM 1 with Pin-Reordering	Leakage Reduction	
total min. leakage (nA) (TL1)	total max. leakage (nA) (TL2)	total leakage (nA) (TL4)	TL1–TL4 / TL1	TL2–TL4 / TL2
530	**645**	**491**	**7%**	**24%**

applied by the MLV generated by ALGORITHM 1. Columns 10–13 show the experiments about the whole version of ALGORITHM 1 that inherent CPV (i.e. CPV_I) is set to (X, X,…, X). The last five columns summarize the reduction percentages. The minimum leakage obtained by our algorithm is averagely less than RSP's minimum and maximum leakages by 8% (third last column) and 20% (last column), respectively. If we ignore the overhead of a CPV controller, 8% and 20% can increase to 11% and 23%, respectively. The CPU time of ALGORITHM 1 is averagely less than RSP by 93%.

From the results show in columns 11 and 12 of TABLE IV, we can draw the conclusion that area and timing overheads of a CPV controller are small. The reason is that a CPV controller is simpler compared with the whole design.

From the last two columns of TABLE IV, we can infer that there is a possibility of additional reduction on the leakage current if the circuit designers can specify a strict CPV_I rather than vector (X, X,…, X) for ALGORITHM 1.

Finally, we compare our results with the results reported by [13] and [18] in the seven common large benchmark circuits that are C880, C1355, C1908, C6288, C7552, i6 and i7. Because the standard cells with different leakage currents are used in [13], [18] and our Leakage Current Solver, we only compare the average reduction percentage of the leakage current rather than the average reduced leakage current. Table V shows the comparison result. In this table, the reduction percentage of the leakage current is as compared to the minimum leakage current obtained from a random search program.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a fast probability-based algorithm for leakage current reduction of digital circuits that operate in the sleep mode. Unlike most other papers that do not count the leakage current overhead of the newborn input vector controller, our algorithm can take this overhead into account. Therefore, in theory our algorithm can more easily find the real optimum solution than the traditional heuristic algorithms. Experimental results show that our technique can reduce the leakage current up to 48% and can find the optimum solutions on 22 out of 26 small MCNC benchmark circuits. Moreover, the CPU time of our technique is averagely less than Random Search Program by 93%.

State encoding, technology mapping, MTCMOS, and gate replacement are feasible techniques for leakage current reduction. We will integrate these techniques into our system in the future. Furthermore, we will study how to reduce the leakage current of a circuit that operates in the normal mode.

TABLE IV
EXPERIMENTAL RESULTS FOR RANDOM SEARCH AND ALGORITHM 1

Benchmark	#PI / #FF	Area (um^2)	CP Delay (ns)	Random Search				Algorithm 1 (CPV_I = (X, X,..., X))					Reduction				
								w/o CPV Controller	w/ CPV Controller								
				#vec-tors	leak. (nA) min. (L4)	leak. (nA) max. (L5)	CPU time (s) (T1)	leak. (nA) (L6)	leak. (nA) (L7)	overhead area	overhead timing	CPU time (s) (T2)	T1–T2 / T1	L4–L6 / L4	L4–L7 / L4	L5–L6 / L5	L5–L7 / L5
C499	41/0	1160	2.9	10K	111	129	12.3	103	107	5.8%	6.0%	0.7	94%	7%	4%	20%	17%
C880	60/0	956	2.5	50K	91	109	65.5	83	87	9.3%	6.8%	0.9	99%	9%	4%	24%	20%
C1355	41/0	1321	2.8	10K	128	143	15.4	115	118	4.6%	6.0%	1.1	93%	10%	8%	20%	17%
C1908	33/0	1126	3.6	10K	106	127	15.2	98	99	3.0%	4.8%	0.6	96%	8%	7%	23%	22%
C3540	50/0	2675	4.0	50K	269	295	43.0	244	248	2.4%	4.2%	9.8	77%	9%	8%	17%	16%
C6288 (Rt=0.2)	32/0	6282	11.2	10K	578	684	535.9	450	453	0.7%	1.5%	31.0	94%	22%	22%	34%	34%
C7552	207/0	5080	4.1	100K	527	567	2144.0	467	484	6.6%	4.1%	65.9	97%	11%	8%	18%	15%
i6	138/0	1101	1.5	100K	103	127	176.9	87	94	14.9%	11.3%	2.2	99%	16%	9%	31%	26%
i7	199/0	1444	1.8	100K	131	157	304.3	115	130	21.6%	9.4%	4.6	98%	12%	0%	27%	17%
i9	88/0	1691	1.6	50K	156	197	216.6	136	143	7.4%	10.3%	5.5	97%	13%	8%	31%	27%
s5378	35/179	5592	2.1	100K	310	332	181.7	286	289	0.9%	8.0%	5.9	97%	8%	7%	14%	13%
s15850	77/538	15320	3.6	100K	806	858	1666.9	727	734	0.9%	4.7%	506.8	70%	10%	9%	15%	14%
Average	–	**3645**	**3.5**	–	**276**	**310**	**448.1**	**243**	**249**	**6.5%**	**6.4%**	**52.9**	**93%**	**11%**	**8%**	**23%**	**20%**

TABLE V
COMPARISON WITH PREVIOUS WORKS
(† Functions of pin reordering and controller synthesis are turned off.)

[18]		MLP [13]		Algorithm 1†	
Leakage Reduction	CPU Time (sec)	Leakage Reduction	CPU Time (sec)	Leakage Reduction	CPU Time (sec)
0.36%	**N/A**	**2.6%**	**17.1**	**3.6%**	**11.3**

REFERENCES

[1] N. Jayakumar and S. P Khatri, "An ASIC Design Methodology with Predictably Low Leakage, Using Leakage-immune Standard Cells," *ISLPED'03* , Aug. 2003, pp. 128–133.

[2] H. Kim and Y. Shin, "Analysis and Optimization of Gate Leakage Current of Power Gating Circuits," *Proc. of ASP-DAC*, Jan. 2006, pp. 565–569.

[3] N. Hanchate and N. Ranganathan, "LECTOR: A Technique for Leakage Reduction in CMOS Circuits," *IEEE Trans. on VLSI Systems*, 2004, pp. 196–205.

[4] Hyo-Sig Won, Kyo-Sun Kim, Kwang-Ok Jeong, Ki-Tae Park, Kyu-Myung Choi and Jeong-Taek Kong, "An MTCMOS Design Methodology and Its Application to Mobile Computing," *ISLPED'03*, August 2003, pp.110–115.

[5] L.Wei, Z. Chen, M. Johnson, K. Roy and V. De, "Design and Optimization of Low Voltage High Performance Dual Threshold CMOS Circuits", *Proc. of DAC*, June 1998, pp 489–494.

[6] B. H. Calhoun, F. A. Honore and A. P. Chandrakasan, "A Leakage Reduction Methodology for Distributed MTCMOS," *IEEE Journal of Solid-State Circuits*, May 2004, pp. 818–826.

[7] S. Sirichotiyakul, T. Edwards, Oh Chanhee, R. Panda, and D. Blaauw, "Duet: an Accurate Leakage Estimation and Optimization Tool for Dual-Vt Circuits," *IEEE Trans. on VLSI Systems*, Apr. 2002, pp. 79–90.

[8] M. Ketkar and S. S. Sapatnekar, "Standby Power Optimization via Transistor Sizing and Dual Threshold Voltage Assignment," *Proc. of ICCAD*, Nov. 2002, pp. 375–378.

[9] D. Lee, H. Deogun, D. Blaauw and D. Sylvester, "Simultaneous State, Vt and Tox Assignment for Total Standby Power Minimization," *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2004, pp. 494–499.

[10] http://www-device.eecs.berkeley.edu/~ptm

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

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

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

[14] L. Yuan and G. Qu. "Enhanced Leakage Reduction Technique by Gate Replacement," *Proc. of DAC*, June 2005, pp. 47–50.

[15] J. P. Halter and F. N. Najm, "A Gate-Level Leakage Power Reduction Method for Ultra-Low-Power CMOS Circuits," *Proc. of IEEE Custom Integrated Circuits Conference, 1997*, pp. 475–478.

[16] D. Lee and D. Blaauw, "Gate Oxide Leakage Current Analysis and Reduction for VLSI Circuits," *IEEE Trans. on VLSI Systems*, Feb. 2004, pp. 155–166.

[17] D. Lee, W. Kwong, D. Blaauw and D. Sylvester, "Analysis and Minimization Techniques for Total Leakage Considering Gate Oxide Leakage," *Proc. of DAC*, June 2003, pp. 175–180.

[18] 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," *Proc. of ICCAD*, 2003, pp. 689–692.

[19] K. Chopra and S. B. K. Vrudhula, "Implicit Pseudo Boolean Enumeration Algorithms for Input Vector Control," *Proc. of DAC*, June 2004, pp. 767–772.

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