A Probabilistic Method to Determine the Minimum Leakage Vector for Combinational Designs

Kanupriya Gulati Nikhil Jayakumar Sunil P Khatri
Department of EE, Texas A&M University, College Station TX 77843.

Abstract

"Parking" a circuit in a minimum leakage state during its standby mode of operation is one of the techniques of reducing leakage power consumption in a circuit. However, the problem of finding this minimum leakage state is NP-hard. In this paper, we present a heuristic approach to determine the input vector which minimizes leakage for a combinational design. Our approach utilizes approximate signal probabilities of internal nodes to aid in finding the minimum leakage vector. We use a probabilistic heuristic to select the next gate to be processed, as well as to select the best state of the selected gate. A fast SAT solver is employed to ensure the consistency of the assignments that are made in this process. Experimental results indicate that our method has very low runtimes, with excellent accuracy, compared to existing approaches.

1. Introduction

Traditionally, dynamic (switching) power has dominated the total power consumption of a VLSI IC. However, due to current scaling trends, leakage power has now become a major component of the total power consumption in VLSI circuits. The leakage current for a PMOS or NMOS device corresponds to the I_{ds} of the device when the device is in the *cut-off* or *sub-threshold* region of operation. The expression for this current [1] is:

$$I_{ds} = \frac{W}{L} I_0 e^{\left(\frac{V_{gs}-V_T-V_{off}}{nv_t}\right)}\left(1 - e^{\left(-\frac{V_{ds}}{v_t}\right)}\right) \quad (1)$$

Here I_0 and V_{off}[1] are constants, while v_t is the thermal voltage (26mV at 300°K) and n is the sub-threshold swing parameter. Note that I_{ds} increases exponentially with a decrease in V_T. This is why a reduction in supply voltage (which is accompanied by a reduction in threshold voltage) results in exponential increase in leakage. This is expected to be a major concern for VLSI design in the nanometer realm [2]. Further, the increasing demand for portable/hand-held electronics has meant that leakage power consumption has received even greater attention. Since these portable devices spend most of their time in a *standby* state (also sometimes called *sleep* state), reducing the leakage power consumption in this *standby* state is crucial to extending the battery life of these designs.

One of the natural techniques for reducing the leakage of a circuit is to gate the power supply using power-gating transistors (also called *sleep* transistors). Typically high-V_T power-gating transistors are placed between the power supply and the logic gates (MTCMOS [3, 4]). In some cases these power-gating transistors are embedded in the logic gates itself [5]. In standby, these power-gating transistors are turned off, thus shutting off power to the circuit in question. Such power-gating techniques can reduce circuit leakages by 2 to 3 orders of magnitude. However, the addition of a power-gating transistor causes an increase in delay of the circuit. Further, the process of waking the circuit up involves a delay (and a power transient), since the supply rails need to reach their stable values before the circuit can operate again.

Increasing V_T via body effect and bulk voltage modulation [6, 7] is another way to reduce leakage power. The leakage current of a transistor decreases with greater applied Reverse Body Bias (RBB). RBB affects V_T through body effect, and sub-threshold leakage has an exponential dependence on V_T as seen in Equation 1. The body effect equation can be written as $V_T = V_T^0 + \gamma\sqrt{V_{sb}}$ where V_T^0 is the threshold voltage at zero V_{sb}.

All the techniques listed above require significant circuit modifications in order to reduce leakage. Another technique, which achieves up to 2 orders of magnitude leakage reduction, is the technique of *parking* a circuit in its minimum leakage state. This technique involves very little or no circuit modification and does not require any additional power supplies. A combinational circuit is *parked* in a particular state by driving the primary inputs of the circuit to a particular value. This value can be scanned in or forced using MUXes (with the standby/sleep signal used as a select signal for the MUX).

Table 1 shows the leakage of a NAND3 gate for all possible input vectors to the gate. The leakage values shown are from a SPICE simulation using the 0.1μ BPTM [8] models, with a VDD of 1.2V.

Input	Leakage(A)
000	1.37389e-10
001	2.69965e-10
010	2.70326e-10
011	4.96216e-09
100	2.62308e-10
101	2.67509e-09
110	2.51066e-09
111	1.01162e-08

Table 1: Leakage of a NAND3 gate

As can be seen from Table 1, setting a gate in its minimal leakage state (000 in the case of the NAND3 gate) can reduce leakage by about 2 orders of magnitude. Ideally, it is desirable to set *every* gate in the circuit to its minimal leakage state. However, this may not be possible due to the logical inter-dependencies of the inputs of the gates. Finding this minimum leakage input vector is an NP-hard problem. Several research efforts have addressed the problem of determining an input vector that minimizes leakage for a design. Our approach falls into this category. The problem of finding a minimal leakage vector can be viewed as one of selecting the state of each gate in the circuit such that the total leakage over all gates is minimized, and the states of each gate in the circuit are logically feasible (i.e. is logically compatible with states of all the other gates). The main feature of our approach is that it is guided by signal probabilities. In other words, the selection of the best candidate gate, as well as the input state to use for that gate, is performed probabilistically. The intuition behind such selections is that they have a high likelihood of resulting in a circuit state which is logically justifiable, while minimizing leakage as well.

The remainder of this paper is organized as follows: Section 2 discusses some previous work in this area. In Section 3 we describe our heuristic method to find the minimum leakage vector (MLV) of a circuit. In Section 4 we present experimental results, while conclusions and future work are discussed in Section 5.

2. Previous Work

The problem of finding the minimum leakage sleep vector for a combinational CMOS gate-level circuit has received some attention recently. In [9], the authors find a minimal leakage vector

[1] Typically $V_{off} = -0.08V$

0-7803-9390-2/06/$20.00 ©2006 IEEE

using random search with the number of vectors used for the random search selected to achieve a specified statistical confidence and tolerance. In [10], the authors reported a genetic algorithm based approach to solve the problem. The authors of [11] introduce a concept called leakage observability, and based on this idea, describe a greedy approach as well as an exact branch and bound search to find the maximum and minimum leakage bounds. The work of [12] is based on an ILP formulation. It makes use of pseudo-Boolean functions which are incorporated into an optimal ILP model and a heuristic mixed integer linear programming method as well. In contrast to these approaches, our approach is a heuristic that uses signal probabilities and leakage values of the gates to help assign values to the nodes in a combinational circuit. In [13, 14], the authors present an MDD [15] based algorithm to determine the lowest leakage state of a circuit. Unlike our method, [14] computes a leakage histogram for the design. The use of MDD based MLV computations limits the applicability of [13] to large designs.

In [16], the authors present a greedy search based heuristic, guided by node controllabilities and functional dependencies. The algorithm used in [16] involves finding the controllability and the controllability lists of all nodes in circuit and then using this information as a guide to choose gates to set to a low leakage state. The controllability of a node is defined as the minimum number of inputs that have to be assigned to specific states in order to force the node to a particular state (based on concepts used in automatic test pattern generation). Controllability lists are defined as the minimum constraints necessary on the input vector to force a node to particular state. The time complexity of their algorithm is reported to $O(n^2)$ where n is the number of cells (gates) in the circuit. However in estimating the complexity of their algorithm, it is not clear if the authors include the time taken to generate the controllabilities and controllability lists of each node in the circuit. While finding the controllabilities can be done fairly easily [17], generating the controllability lists can be more involved. In our approach we do not compute node controllabilities or their controllability lists. We compute signal probabilities instead. The algorithm for this is detailed in section 3

In [18], the authors express the problem of finding a minimum leakage vector as a satisfiability problem and use an incremental SAT solver to find the minimum and maximum leakage current. While their approach worked well for small circuits, the authors report very large runtimes for large circuits. The authors therefore suggest using their algorithm as a checker for the random search suggested in [9]. Our approach can handle larger circuits with low run-times and good accuracy, as shown in Section 4.

3. Our Approach

The outline of our methodology for selecting the input vector that minimizes circuit leakage is as follows:

- First, we compute signal probabilities for all nodes in the design, assuming that all inputs have a signal probability of 0.5[2]. These probabilities are heuristically adjusted for inaccuracies arising from reconvergent fanouts.
- Next, we select the best candidate gate whose leakage we would like to set in a given iteration. This is performed by selecting the gate that is probabilistically most likely to result in the largest leakage reduction.
- For the gate thus selected, we next assign its best state, such that the leakage of the selected gate is probabilistically minimized. All other gates in the circuit which are newly implied by the state just selected are accounted for while making this decision.
- We test if the logic values that were set to 1 or 0 during this iteration are satisfiable, by calling a Boolean Satisfiability solver. The SAT solver is called every p iterations to reduce the runtime. If the circuit is unsatisfiable, we undo the assignments of the last p iterations, and find the iteration that caused the circuit to become unsatisfied. After making a different selection for that iteration, we proceed as before.
- After any iteration, gate probabilities are adjusted, to account for the nodes that were newly assigned fixed logic values.
- A fixed number of passes are made for the circuit, with the above steps being applied successively. Each pass is more "lenient" in setting a node to a logic value v when its signal probability deviates from v. The last pass is most lenient, allowing any deviation from v to be accepted.

Algorithm 1 describes the pseudocode for our approach for computing the MLV for a combinational network η.

Algorithm 1 Pseudocode of Minimum Leakage Vector Algorithm

```
compute_minimum_leakage_vector(η, p){
compute_signal_probabilities(η)
platinumvalues ← Φ
for i = 1; i ≤ k; i++ do
  goldvalues ← Φ
  iteration = 1
  (G = find_best_gate(η)
  if (G is not marked visited) then
    (S = find_best_leakage_state(G, η)
    if S satisfies m_i then
      goldvalues ← goldvalues ∪ S ∪ get_implications(S)
      propagate probabilities in TFO of goldvalues nodes
    end if
    if iteration is a multiple of p OR all inputs assigned/implied then
      if goldvalues are satisfiable then
        if all inputs assigned then
          exit
        end if
        platinumvalues ← platinumvalues ∪ goldvalues
      else
        goldvalues ← platinumvalues
      end if
    end if
  end if
  iteration += 1
end for
}
```

3.1 Computing Signal Probabilities

The algorithm *compute_minimum_leakage_vector*(η) begins by computing signal probabilities for all nodes in the network η. The inputs are assumed to have probabilities of 0.5, and these probabilities are propagated throughout the circuit[3]. After the initial pass of propagation, we heuristically adjust for reconvergent fanouts. The heuristic for probability adjustment in the presence of reconvergence is illustrated in Figure 1.

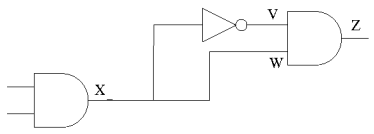

Figure 1: Adjusting Probabilities for Reconverging Nodes

Suppose a node X, with a statically computed probability of P_X reconverges at Z. Then we set the probability of X to 1 and 0, and find the probabilities of the inputs to the reconvergent gate (V and W). Suppose the probabilities of V (W) are V_1 (W_1) and V_0 (W_0) respectively, when X is set to 1(0). In this case, the new probability of Z is $P_Z^{new} = \frac{V_0 \cdot W_0 + V_1 \cdot W_1}{2}$.

[2]In case of sequential circuits, we could utilize the probabilities of the signals at the outputs of memory elements instead.

[3]If the input i of an n-input AND gate has probability p_i, then the output has probability $\Pi_i p_i$. Likewise, for an OR gate, the output has probability $1 - \Pi_i(1 - p_i)$. The probabilities of other gates can be found in a similar fashion

From this we compute the adjustment factor for the probability of Z, as follows:

$Adjustment(Z) = \frac{(P_Z^{new} - P_Z)}{P_Z}$

In future updates of the probability of the node Z, suppose the statically computed probability of node Z is $P_Z^{modified}$. In that case, the final adjusted value of the probability of node Z is

$P_Z^{adj} = (P_Z^{modified}) \cdot (1 + Adjustment(Z))$.

In other words, $Adjustment(Z)$ is computed once, and utilized to adjust the statically computed values of the probability of node Z, each time it is modified due to other assignments in the circuit.

In the example of Figure 1, $Adjustment(Z) = -1$. Therefore, $P_Z^{adj} = 0$ each time the probability of Z is modified. This is reasonable, given that the output Z is logically 0.

If an adjustment of the probability of a node results in its probability becoming higher than P_{adj} (lower than $1 - P_{adj}$), then the probability of the node is capped at P_{adj} $(1 - P_{adj})$ respectively.

3.2 Finding Best Leakage Candidate

Once signal probabilities are computed, we next select the best candidate gate whose input state we would like to finalize. Gates are ranked by the probabilistic criterion described below:

$C = \frac{\sum(p_i \cdot l_i)}{\sum(p_i)}(l_i^{max} - l_i^{min})$

Here, p_i is the probability that the gate is in state i. By "state", we mean a complete assignment of the inputs (and outputs) of the gate. The quantity l_i is the leakage of the state i. The value l_i^{max} (l_i^{min}) is the maximum (minimum) leakage value of this gate. The gate with the maximum value of C is selected. In other words, this criterion selects gates that have a high probability of being in a high-leakage state. The last term in the expression for C ensures that gates with large leakage ranges are favored, since they offer potentially greater optimization flexibility. The gate that maximizes C is selected preferentially over others.

3.3 Finding Best Leakage State for Selected Gate

Suppose a gate G was selected by the previous step. We now want to assign it a state such that its leakage is minimized. This is done by applying the probabilistic criterion L below. Note that all gates other than G whose states become fully assigned[4] on account of implying the current state of G, are included in the computation of L. Let the number of such gates be n. The value of probabilistic leakage in the numerator of L is normalized with respect to the number of such gates, and is computed as follows:

$L = \frac{\sum_j (d_j \cdot l_j)}{n}$

The state of gate G that minimizes L is preferentially selected over others. Here d_j is the deviation of the values assigned to the gate inputs from their probabilistic values. For example, consider an AND gate with inputs a and b with probabilities 0.1 and 0.7 respectively. If inputs a and b are logic 1 and logic 0 respectively, then the deviation is $(|1 - 0.1|)(|0.7 - 0|)$.

In order to bias the state selection towards assignments with lower leakage, the deviation is incremented by a value β. Likewise, in order to bias the state selection towards those with lower deviation, we increment l_j by a fixed value γ. Therefore, the modified value of L that is used is

$L = \frac{\sum_j (d_j + \beta) \cdot (l_j + \gamma)}{n}$

3.4 Accepting Leakage States and Endgame

The state selected from the previous step is now implied throughout the circuit. The resulting values are referred to as golden values. The deviations of the resulting implications are now checked against a margin value m_i. If any deviation is greater than m_i, then the assignment to gate G is discarded. Initially, m_i is set to a small value, and with increasing iteration i, it is relaxed. This is in an attempt to get closer to a global minima, by a more careful selection of states in early iterations. We perform $k = 3$ iterations in our experiments.

[4]A gate is said to be *fully assigned* if all its inputs are assigned to specific logic values

Once the new implications are computed, the implied nodes' probabilities are adjusted to reflect the freshly computed implications. If a node is set to a logic 1, then its probability is set to (1-α), while a node which is set to logic 0 has its probability updated to α.

For every p gates selected (or if all primary inputs have been assigned or implied), we test if the golden values are satisfiable (this test is done by invoking the BerkMin [19] satisfiability solver). If so, then all golden values are designated as new platinum values, never to be modified in the future. If the golden values are satisfiable, and all inputs are assigned, then the algorithm exits. If the golden values could not be satisfied, then we roll back the golden values, by copying the last set of platinum values into the set of golden values. For up to the next p iterations, we call the satisfiability solver after each new state assignment. This is in an attempt to locate which of the last p assignments caused the unsatisfiability condition to occur. Once this state is identified, we again revert to calling the satisfiability solver after every p state assignments. If the satisfiability solver returns an unsatisfiable condition for a certain state s assigned at a particular gate g, then we never try assigning s to g again.

4. Experimental Results

We performed extensive experiments to validate our method and compare its results to the exact minimum circuit leakage values. When it was not possible to find the exact minimum circuit leakage values, we found the minimum leakage value over a large number of input vector samples. In all our experiments, we utilized a value of $k = 3$ iterations. The 3 sets of parameter sets (M1, M2 and M3) that we utilized for our experiments are described in Table 2. These are referred to as *methods* in the sequel. The value of p used was 1, but it can be increased for less accurate but faster invocations of the algorithm. The values reported in Table 2 were found after extensive experimentation with many circuits.

Method	m_1	m_2	m_3	β	γ	P_{adj}	α
M1	0.6	0.96	1	0.5	50	0.95	0.95
M2	0.6	0.96	1	5	10	0.95	0.95
M3	0.4	0.96	1	0.1	100	0.9	0.9

Table 2: Parameters used in our Experiments

Method M1 and M2 utilizes a value of m_1 of 0.6. As a consequence, we expect to set more gates to platinum values in the first iteration. These methods are designed to reduce the number of gates discarded due to margin violations. Among these methods, M1 has a higher γ value, and therefore biases the state selection towards states which have smaller deviations. On the other hand, M2 has a higher β value, and as a consequence, state selection favors states with lower leakage. Method M3 has a smaller m_1 value, and therefore tends to reject gates due to margin violations. It is biased towards state selections which have smaller deviations.

Using these three methods, we first compared the results of our method with those of an exhaustive evaluation of leakages. This was performed for small examples, and results are reported in Table 3. The minimum leakage value returned by our method (Column 4), along with the exact maximum (Column 2) and minimum (Column 3) leakages are shown in this table. Further, we report a figure of merit R in Column 5.

$R = \frac{Our\ min\ leakage - Exact\ min\ leakage}{Exact\ max\ leakage - Exact\ min\ leakage}$

The values of the maximum and minimum range of leakages are computed based on an exhaustive simulation of the circuit. Ideally, R should be 0. Runtimes for our method are reported in Column 7, while the method utilized is reported in Column 6.

Note that the figure of merit R is a more rigorous metric for comparing the effectiveness of any MLV determination technique. In the prior approaches to the MLV determination problem, the figure of merit utilized was

$R_{old} = \frac{Heuristic\ min\ leakage - Exact\ min\ leakage}{Exact\ min\ leakage}$

Based on Table 3, the average value of R for our method was

about 0.125. For our method, the average value of the previously utilized figure of merit is about 0.053.

Table 3 shows that the runtimes for our method are very small, with a good figure of merit for the method. Given that the runtimes are very small, *we can afford to apply all three methods (M1, M2 and M3), and choose the best result among the three. In general, we may try several methods and select one that yields the vector with the smallest leakage.*

We also tested our method on larger circuits. The results of this experiment are shown in Table 4. The columns in this table are as in Table 3, with the exception that exact leakage values are not computed in this table. Instead, the minimum and maximum leakage found over 10,000 random vectors is shown in Table 4. According to [9], this statistically yields a greater than 99% confidence that we will obtain the lowest 0.5% leakage vector.

Table 4 shows that our method produces MLVs with very low errors, with extremely small runtimes. From [12], for the previously reported methods of [12], [20] and [16], the errors were respectively 5.3%, 3.7% and 10.4% (using the R_{old}[5] metric, for which our method results in an error of 3.7%). Further, the runtimes for our method are significantly smaller than those of [20].

Circuit	Low	High	Our Low	R	R_{old}	Meth.	Time (s)
C17	12.47	30.15	12.47	0.000	0.000	M2	0
cm138a	70.80	109.86	71.63	0.021	0.012	M1	0.01
cm151a	129.55	167.57	135.75	0.163	0.048	M2	0.02
cm152a	65.95	99.07	75.54	0.290	0.145	M3	0.01
cm42a	90.19	111.23	90.73	0.026	0.006	M1	0
cm82a	87.22	101.16	91.37	0.298	0.048	M2	0.02
cm85a	142.25	217.08	162.08	0.265	0.139	M1	0.06
decod	185.97	237.43	185.97	0.000	0.000	M3	0.08
majority	32.41	44.55	32.41	0.000	0.000	M2	0.01
t	17.48	33.72	20.7	0.198	0.184	M2	0
AVG				0.1261	0.053		

Table 3: Exhaustive and Estimated Leakages for Small Circuits

Circuit	Low	High	Our Low	R	R_{old}	Meth.	Time (s)
apex6	2738.57	3085.62	2609.550	-0.372	-0.047	M3	81.86
b9	328.16	554.04	362.11	0.150	0.103	M2	0.47
C1908	2353.45	2833.71	2473.570	0.250	0.051	M2	42.73
C2670	3430.12	3880.88	3459.29	0.065	0.009	M2	141.31
C3540	5268.64	5918.25	5402.009	0.205	0.025	M1	296.19
C432	841.13	1074.19	893.27	0.224	0.062	M2	1.78
C499	1748.97	1942.81	1775.1	0.135	0.015	M2	20.18
c8	507.41	829.42	552.24	0.139	0.088	M3	1.46
cc	190.44	402.85	203.87	0.063	0.071	M1	0.12
cht	735.31	1066.45	777.480	0.127	0.057	M3	3.39
cm150a	257.87	324.84	264.81	0.104	0.027	M3	0.21
cm162a	157.88	228.34	173.2	0.217	0.097	M1	0.04
cm163a	151.72	213.44	165.62	0.225	0.092	M1	0.05
cmb	119.87	203.63	130.61	0.128	0.090	M1	0.03
comp	523.47	691.71	563.31	0.237	0.076	M3	1.03
count	483.50	624.68	488.97	0.039	0.011	M3	0.81
cu	159.71	276.39	184.16	0.210	0.153	M3	0.14
example2	1079.71	1396.58	998.44	-0.256	-0.075	M3	9.49
frg1	362.57	499.46	377.66	0.110	0.042	M3	0.6
i1	124.17	234.66	127.47	0.030	0.027	M1	0.05
i3	705.68	823.34	672.36	-0.283	-0.047	M1	2.94
i4	624.79	951.79	539.9	-0.260	-0.136	M3	4.29
i5	1090.40	1417.71	909.120	-0.554	-0.166	M2	9.44
lal	357.52	687.27	389.02	0.096	0.088	M3	0.94
mux	287.66	400.43	304.31	0.148	0.058	M1	0.22
parity	213.38	271.10	221.56	0.142	0.038	M2	0.08
pcle	219.48	316.58	229.22	0.100	0.044	M1	0.07
pcler8	307.10	397.61	310.9	0.042	0.012	M3	0.18
pm1	92.63	285.76	95.66	0.016	0.033	M3	0.07
rot	2294.48	2658.33	2321.8	0.075	0.012	M3	63.35
tcon	139.77	190.54	139.77	0.000	0.000	M1	0.04
ttt2	857.95	1222.36	919.51	0.169	0.072	M2	3.43
unreg	473.89	615.77	497.68	0.168	0.050	M1	0.51
x1	1156.81	1708.28	1273.85	0.212	0.101	M1	10.65
x3	3219.92	4274.29	3528.949	0.293	0.096	M1	167.64
x4	1690.86	2471.53	1919.870	0.293	0.135	M3	30.27
AVG				0.0746	0.037		

Table 4: Leakages for Large Circuits

5. Conclusions

We have developed a probabilistic method to perform input vector assignment for leakage minimization in a combinational circuit. We start by computing signal probabilities throughout the circuit. These probabilities are used to guide the selection of the next gate to assign. The selected gate is the one with the probabilistic highest leakage value. Once this gate is selected, it is assigned a state, again in a manner which probabilistically minimizes its leakage. The implications induced by such a state selection are computed. A satisfiability solver is invoked, to validate the state selection before our algorithm commits to this assignment. The algorithm terminates when all inputs have been assigned or are implied.

The method is fast, flexible and provides accurate results. On average, for small examples, our method found minimum leakage values which were 5.3% from the minimum circuit leakage. For larger examples, it was impractical to compute the minimum circuit leakage exactly. We computed our statistics on the basis of running 10,000 samples of circuit leakage computation. For these examples, our method produces MLVs with leakage within 3.7% from the minimum. The runtimes of our method are much lower than existing techniques which produce results of similar quality.

[5]Although the R metric is more rigorous, our comparisons to existing approaches utilize the R_{old} metric since these approaches utilize the R_{old} metric.

References

[1] "BSIM3 Homepage." http://www-device.eecs.berkeley.edu/~bsim3/arch_ftp.ht

[2] "The International Technology Roadmap for Semiconductors." http://public.itrs.net/, 2003.

[3] J. T. Kao and A. P. Chandrakasan, "Dual-threshold voltage techniques for low-power digital circuits," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 1009–1018, Jul 2000.

[4] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada, "1-v power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *IEEE Journal of Solid-State Circuits*, vol. 30, pp. 847–854, Aug 1995.

[5] N. Jayakumar and S. Khatri, "An ASIC design methodology with predictably low leakage, using leakage-immune standard cells," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 128–133, Aug 2003.

[6] H. Kawaguchi, K. Nose, and T. Sakurai, "A super cut-off CMOS (SCCMOS) scheme for 0.5-v supply voltage with picoampere stand-by current," *IEEE Journal of Solid-State Circuits*, vol. 35, pp. 1498–1501, Oct 2000.

[7] 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 Transactions on Electron Devices*, vol. 44, pp. 414–422, Mar 1997.

[8] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proc. of IEEE Custom Integrated Circuit Conference*, pp. 201–204, Jun 2000. http://www-device.eecs.berkeley.edu/ ptm.

[9] J. Halter and F. Najm, "A gate-level leakage power reduction method for ultra low power cmos circuits," in *Proceedings of CICC*, pp. 475–478, 1997.

[10] Z. Chen, M. Johnson, L. Wei, and W. Roy, "Estimation of standby leakage power in CMOS circuit considering accurate modeling of transistor stacks," in *International Symposium on Low Power Electronics and Design*, pp. 239–244, 1998.

[11] M. 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, June 1999.

[12] F. Gao and J. Hayes, "Exact and heuristic approaches to input vector control for leakage power reduction," in *Proceedings, International Conference on Computer-aided Design*, pp. 527–532, Nov 2004.

[13] K. Chopra and S. Vrudhula, "Implicit pseudo Boolean enumeration algorithms for input vector control," in *Proceedings, Design Automation Conference*, (San Diego), pp. 767–772, June 2004.

[14] K. Gulati, N. Jayakumar, and S. Khatri, "An Algebraic Decision Diagram (ADD) based technique to find leakage histograms of combinational designs," in *Proceedings, International Symposium on Low Power Electronic Design (ISLPED)*, (San Diego, CA), August 2005.

[15] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic decision diagrams and their applications," *Formal Methods in Systems Design*, vol. 10, no. 2/3, pp. 171–206, 1997.

[16] R. Rao, F. Liu, J. Burns, and R. Brown, "A heuristic to determine low leakage sleep state vectors for CMOS combinational circuits," in *Proceedings, International Conference on Computer-aided Design*, pp. 689–692, Nov 2003.

[17] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[18] F. Aloul, S. Hassoun, K. Sakallah, and D. Blauuw, "Robust SAT-based search algorithm for leakage power reduction," in *Proceedings, Power and Timing Models and Simulation (PATMOS)*, 2002.

[19] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT-solve," in *Proceedings, Design Automation and Test in Europe (DATE) Conference*, pp. 142–149, 2002.

[20] S. Naidu and E. Jacobs, "Minimizing stand-by leakage power in static CMOS circuits," in *Proceedings, Design Automation and Test in Europe (DATE) Conference*, pp. 370–376, March 2001.