An Algorithm to Minimize Leakage through Simultaneous Input Vector Control and Circuit Modification

Nikhil Jayakumar
nikhil_AT_ece_DOT_tamu_DOT_edu

Sunil P Khatri
sunilkhatri_AT_tamu_DOT_edu

Department of Electrical & Computer Engineering,
Texas A&M University, College Station TX 77843.

Abstract

Leakage power currently comprises a large fraction of the total power consumption of an IC. Techniques to minimize leakage have been researched widely. In this paper, we present an approach which minimizes leakage by simultaneously modifying the circuit while deriving the input vector that minimizes leakage. In our approach, we selectively modify a gate so that its output (in sleep mode) is in a state which helps minimize the leakage of other gates in its transitive fanout. Gate replacement is performed in a slack-aware manner, to minimize the resulting delay penalty.

1 Introduction

Traditionally dynamic (switching) power has dominated the total power consumption of an IC. However, due to current scaling trends, leakage power has now become a major component of the total power consumption in VLSI circuits. Further, the increasing demand for portable/handheld electronics has meant that leakage power 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 such products.

One of the techniques used to minimize leakage 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 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). This is frequently referred to as *input vector control*. In this paper we extend this technique to achieve control over the leakage of a circuit at a finer granularity. The leakage reduction technique discussed in this paper is orthogonal to other circuit level leakage reduction approaches such as MTCMOS and others that statically or dynamically change the V_T of the devices.

The remainder of this paper is organized as follows: The motivation for this work is described in Section 2. Section 3 discusses some previous work in this area. In Section 4 we describe our method to minimize leakage in a circuit through simultaneous input vector control and circuit modification. In Section 5 we present experimental results, while conclusions are discussed in Section 6.

2 Motivation

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 [3] models at 1.2V.

Input	Leakage(A)
000	1.37e-10
001	2.70e-10
010	2.70e-10
011	4.96e-09
100	2.62e-10
101	2.68e-09
110	2.51e-09
111	1.01e-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. This leakage reduction is attributed to the *stack effect*, according to which having as many *off* transistors in series as possible minimizes leakage. While it is desirable to set every gate in a circuit to its minimal leakage state, it may not be possible

978-3-9810801-2-4/DATE07 © 2007 EDAA

do so due to the logical inter-dependencies of the inputs of the gates. Even if the individual gates have a wide range of leakage values, that does not mean the circuit that uses these gates will have a wide range of leakage values as well. For example if a NAND3 gate and a NOR3 gate in a circuit share inputs, the leakage of the NAND3 is minimum when all the inputs are set to logic 0, but to get the NOR3 gate into its minimum leakage state requires all the inputs to be set to logic 1. Due to constraints such as these, we are limited in terms of the leakage reduction that we can achieve by using just vector control at the primary inputs. In order to exploit the stack effect better, we need a technique that offers more freedom in setting the inputs at each gate. Herein lies the contribution of this paper. Gate leakage currents can also contribute to the total leakage of a gate. While the contribution of gate leakage may affect the table of leakage values for each input vector for a gate, our algorithm is agnostic to this and only requires a reliable estimate of leakage currents of a gate for different input vectors.

3 Previous Work

Traditionally, input vector control has involved using MUXes or scan-chains to control primary input values of a circuit during *standby*. We extend this idea further by modifying the circuit such that we are not restricted to controlling just the primary inputs, but can also control the internal nodes of a circuit. While the idea of adding control points is similar to what is expressed in [1, 2], we allow a greater degree of freedom. In [1, 2], the authors insert either AND or OR gates to set the logic value of a particular line during *standby*. We, on the other hand, allow one input going to 2 or more different gates to be split (using pass-gate MUXes), so that each fanout can be set to different values during *standby*. *This ability provides significantly more opportunities to control internal nodes and minimize leakage.* Also in [1, 2], the authors use a SAT based algorithm to find control points and to minimize leakage. The accuracy of the algorithm is dependent on the number of quantization levels of leakage values. However, with a higher number of quantization levels the runtime also increases. The algorithm we use has significantly lower complexity, and involves a linear pass of the circuit. In [8], a technique is presented which involves gate replacement. However, in [8] a gate G is replaced by a different gate G' to only reduce the leakage of gate G, *not control internal nodes*. The authors of [4], improve on the implementation of [8] in terms of both leakage improvement and runtime of the gate replacement algorithm. Previous approaches to minimize leakage through vector control and gate replacement [1, 2, 8, 4], have a delay penalty to get a reasonable leakage reduction. In our approach, we get a significant leakage reduction (as shown in Section 5) with *no delay penalty*.

4 Our Approach

One of the sources of our flexibility in controlling internal nodes of a circuit stems from the fact that we define several different variants of each gate in the library. While it may be argued that the creation of different variants of a cell can be time consuming and expensive, it should be noted that this step is done only once up-front. An example of the different variants is shown in Figure 1. In the $snglmx$ type of variant, a MUX is placed at the output of a regular gate. There are two type of $snglmx$ gates, $snglmx_0$ and $snglmx_1$. The $snglmx_0$ gates have a weak pull-down device at the output of the MUX. A $snglmx_0$ variant is used as a replacement for a gate when the output of a gate G is logic 1, but some gates in the fanout of G require a logic 0 to get into a low leakage state. Similarly, $snglmx_1$ gates have a weak pull-up device at the output of the MUX. A $snglmx_1$ variant is used when the output of a gate G is logic 0, but some gates in its fanout require a logic 1 to get into a low leakage state. Note that the $snglmx$ type of variants are dual output gates and hence offer the most flexibility of 'splitting' internal signals.

There can be situations when the all the gates in the fanout of the gate in question need a value that is complementary to what is generated at the output of a gate. For such cases we have a type of variant called the $sngl1out$ variant. This type of variant has only 1 output and is similar to the structure discussed in [2]. We define 2 types of $sngl1out$ variants, $sngl1out_0$ and $sngl1out_1$. The $sngl1out_0$ uses a PMOS sleep transistor to cut-off the PMOS stack of the gate (marked as sleep cut-off in Figure 1) and a weak NMOS pull-down device (marked as sleep bypass in Figure 1) to pull down the output. This variant is used when the output of a gate is high in the standby state, while all the gates in the fanout require a logic low value to get into a low leakage state. Similarly the $sngl1out_1$ uses a NMOS sleep transistor to cut-off the NMOS stack of the gate and a weak PMOS pull-up device to pull up the output. This variant is used when the output of a gate is low in the standby state, while the gates in the fanout require a logic high value to get into a low leakage state. Note that while the $snglmx$ type of variant worsens both output rise and output fall delays, the $sngl1out$ worsens delay for either only the rise or only the fall transition and can actually speed up the opposite transition. *In [2], the authors take advantage of this fact and assume the delay of such a gate to be the average of the rise and fall delays. This assumption can lead to inaccuracies in the timing analysis.* In this paper, we account for the rise and fall delays separately.

Due to the introduction of sleep devices, the delay of the $sngl1out$ gates is larger than the regular cells (for one transition). Similarly, the $snglmx$ variants also suffer a delay due to the pass gate MUX at the output. Since we have tim-

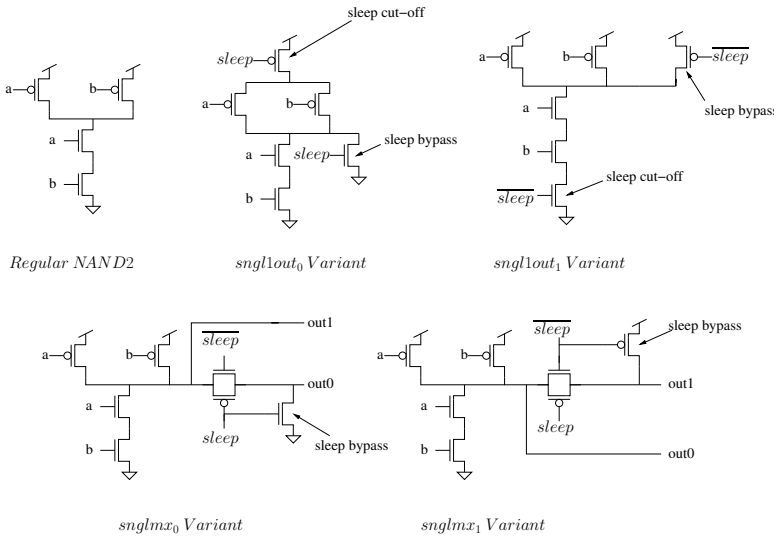

Figure 1. Some Variants of a NAND2 gate

ing constraints, this delay limits the flexibility of the gate replacement algorithm. To enhance the flexibility of the algorithm and give it more degrees of freedom, we also create larger cells that we call dbl cells. We create $dblmx$ and as well as $dbl1out$ variants. Their structure and purpose is the same as their sngl counterparts except that they use larger device sizes ($\leq 2\times$ of their sngl counterparts). They are sized such that their delays are closer to the delays for regular gates.

All these variants are crucial to our approach and help provide enough flexibility to our algorithm to reduce the leakage of a given circuit while making sure there is *no* delay penalty. The details of the algorithm are explained in Section 4.1.

4.1 The Gate Replacement Algorithm

Before we use the gate replacement algorithm, we first characterize our library of cells (including the variants) using SPICE [6], and generate a file in the GENLIB format from the characterized data. In the GENLIB format each pin of a gate is associated with an intrinsic delay component well as a load dependent component for both rise and fall times separately. Also included in the genlib file is the load capacitance of each input pin.

Our algorithm takes as input a netlist of gates in levelized order. We first perform a static timing analysis on this netlist to find the Arrival Times (ATs) and Required Times (RTs) at all nodes in the circuit. We use the cell characterization data for our static timing analysis. We assume that for gates driven by primary inputs, the primary input can be split to get the desired logic value at the inputs of these gates. Once the logic values of the inputs to the 0^{th} level of gates (the gates with only primary inputs as the inputs) has been fixed, we propagate these values forward to the next level. Next, we pick a gate G from the first level. Lets say the output of the gate is a signal g. We then look through the list of gates in the fanout of G and find the input that gives minimum possible leakage. From this we get the logic value required of g. For example, if one of these fanout gates is a 2 input gate H and assuming that one of its inputs is set to 1 due to another gate J, we would pick the minimum leakage from the following set of input vectors (11, 10). Thus we get the value of g required to get this 2 input gate H in its minimum possible leakage state. Note that initially, we assume all possible input vectors are possible at each gate (i.e. we would consider all vectors 00, 01, 10 and 11 to get the minimum possible leakage vector). This step of finding the best value of g is done for all fanouts of G. If we need to set the value of g to 0 for some fanouts and to 1 in others (which would happen in situations where the signal is an input to a nand gate and a nor gate), then we see if we can replace the gate G with its $snglmx$ variant. We first estimate the leakage savings (if any) of doing this replacement. The presence of the MUX and the weak pull-up/ pull-down used in the $snglmx$ variant is a source of additional leakage. However, this increase could be outweighed by the leakage savings at the gates in the fanout of G. We estimate the difference and if there are savings, we then test if replacing G with a $snglmx$ variant causes timing violations. If there are timing violations, we attempt to use a $dblmx$ variant. Again we first check for leakage savings and if there are savings in leakage, we then check for timing violations. When checking for timing violations due to replacing G with a gate G',

Algorithm 1 replaceGateForMinLkg

replaceGateForMinLkg (levelized_netlist, genlib_data, allowed_slack)
find AT at all nodes
find RT at all nodes
set all gates at first level to minimum leakage state
for $(i = 1; i <= maxLevel\_of\_Ckt; i++)$ **do**
  for $(j = 1; j <= num\_of\_gates\_at\_Level(i); j++)$ **do**
    $G = G(j)$; pick a gate G from the gates at level i
    g = output signal of G
    find suggestedVal of g for all fanout(G)
    if all suggestedVal = 0 and logic value of g = 1 **then**
      $G_{new} = sngl1out_0$ variant of G
      CheckIfReplaceable(G, G_{new})
    else if all suggestedVal = 1 and logic value of g = 0 **then**
      $G_{new} = sngl1out_1$ variant of G
      CheckIfReplaceable(G, G_{new})
    else
      $G_{new} = snglmx_0$ variant of G
      CheckIfReplaceable(G, G_{new})
    end if
  end for
end for

Algorithm 2 CheckIfReplaceable

CheckIfReplaceable (G,G_{new})
Check if G can be replaced by a sngl variant
if G can be replaced by sngl variant of G reduction in leakage and satisfying timing **then**
  replace G with the sngl variant
else if G can be replaced by dbl variant of G with reduction in leakage and satisfying timing **then**
  replace G with the dbl variant
end if

we first propagate new RTs at the gate G to its fanins. Also, note that replacing G implies changes in the capacitance seen by the gates in the fanin of G. We recalculate the AT of the gates in the fanin of G. If the new AT is greater than the new RT, then we don't replace G with G'. If there is no timing violation (there is enough slack) and there are possible savings in leakage, then we go ahead and replace the gate G with its $dblmx$ variant. We follow a similar procedure if all the fanouts of G require the same value at g for minimum leakage. If this value required is the same as the value at g due to fixing the logic values at the inputs of G, then we don't need to replace the gate. If however, these value differ, then we attempt to first replace the gate with its $sngl1out$ variant. If such a replacement does not save leakage current, then we don't replace the gate G and move on to the next gate in the netlist. If such a replacement does not work due to timing slack reasons, we then see if a $dbl1out$ variant of G would help without sacrificing power or timing. In this way we traverse the netlist in levelization order from primary inputs to primary outputs and replace gates as we move along to reduce leakage *while guaranteeing that there are no timing slack violations*.

5 Experimental Results

We performed extensive experiments to validate our method and compare its results to the minimum circuit leakage values. If the circuit input space was small we found the exact minimum leakage through exhaustive simulation, if not we simulated the circuit for 10000 random vectors to find the minimum leakage(as suggested in [5]). We assumed a library with the following basic cells: INV1X, INV2X, NAND2, NAND3, NAND4, NOR2, NOR3. The circuits for our simulations are from the ISCAS85 and MCNC91 benchmark suites. We first performed a technology independent synthesis on these circuits in SIS [7] using *script.rugged*.

In Table 2, column 2 and column 3 show the minimum leakage current (in nA) for the original circuit and for the circuit modified by our algorithm, respectively. The % decrease in leakage current is shown in column 4. The decrease in leakage current is 29.18% on average. *Note that this is the leakage decrease compared to the leakage obtained by applying input vector control alone*. The critical delays (in ps) for the original and the modified circuit are shown in columns 5 and 6 respectively. Column 7 give the % decrease in critical delays of the modified circuit. We conjecture that one of the reasons for the delay decreasing is due to the fact that when the algorithm can't choose a $sngl$ variant due to timing issues, it chooses a dbl variant and this can cause a decrease in the delay. Also, as mentioned in Section 4, while the delay of one type of transition gets worse in the *sngl1out* variants, the delay of the opposite transition is sped up slightly. The last column of Table 2, gives the runtimes of the algorithm. The algorithm is currently implemented in PERL and was run on an Intel Pentium 4 with 2GB of RAM, running Linux Fedora Core 3. The runtimes are expected to improve substantially when the algorithm is implemented in a compiled language such as C/C++. Our algorithm assumes that there are MUXes at the primary inputs. They help ensure that all 0^{th} level gates can be set independently into their low leakage state. For a fair comparison, we give the same flexibility (ability for the inputs of each of the 0^{th} level gates to be set independently) when finding the minimum leakage vector for the original circuit.

In Table 3, the area penalty associated with using our algorithm is given. Note that this table refers to only the active

Ckt.	Min Lkg Original(nA)	New Min Lkg(nA)	% Lkg Decr	Original Delay	New Delay	% Delay Incr	Runtime (s)
alu2	1251.72	1022.44	-18.32	1460.70	1422.16	-2.64	5.53
alu4	2598.14	2094.99	-19.37	1755.99	1753.09	-0.17	21.16
apex6	2743.08	1753.82	-36.06	739.94	739.93	-0.00	20.03
apex7	812.72	592.88	-27.05	704.11	704.11	0.00	2.89
C1355	2003.61	1697.87	-15.26	930.41	930.23	-0.02	7.8
C432	584.46	449.93	-23.02	1110.89	1110.89	0.00	1.03
C880	1375.73	977.07	-28.98	1803.93	1718.75	-4.72	6.12
C1908	1909.95	1548.12	-18.94	1489.95	1488.61	-0.09	10.1
C3540	4079.92	3126.00	-23.38	1870.95	1870.63	-0.02	51.89
C6288	13020.10	12011.39	-7.75	5651.08	5637.02	-0.25	695.85
dalu	3293.89	2378.24	-27.80	1506.29	1504.32	-0.13	42.75
des	15218.02	12013.16	-21.06	3021.52	2470.33	-18.24	655.38
i10	8738.32	6318.98	-27.69	2549.68	2499.43	-1.97	238.13
i1	158.38	102.96	-35.00	353.61	353.21	-0.11	0.11
i2	372.66	98.72	-73.51	392.98	392.98	0.00	0.51
i3	323.05	60.13	-81.39	182.46	182.46	0.00	0.98
i6	1907.06	1650.16	-13.47	1080.10	1080.10	0.00	5.5
i7	2499.20	1973.08	-21.05	1088.31	1088.31	0.00	10.38
i8	3805.49	2321.63	-38.99	1591.76	1297.01	-18.52	38.62
i9	2552.20	1440.26	-43.57	1651.78	1618.21	-2.03	15.87
t481	2915.54	2409.63	-17.35	901.69	838.36	-7.02	28.21
too_large	1034.72	796.34	-23.04	680.24	677.89	-0.35	4.09
Avg			-29.18			-2.56	84.68

Table 2. Leakage, Delay Improvements Using Our Approach, Runtimes

area. Column 2 of the table shows the area of the original circuit. Column 3 and column 4 of the table give the total area and the overhead respectively of the modified circuit including the area of the sleep cut-off transistors used in the $sngl1out$ and the $dbl1out$ type of gates. The active area of these sleep cut-off transistors is given in column 5. Column 6 (which is obtained by subtracting column 5 from column 3) and column 7, report the area and overhead respectively of the modified circuit *excluding* the sleep-cut-off transistors. On average, the total active area overhead including the sleep cut-off transistors is about 23.6%. However, the active area overhead excluding the sleep cut-off transistors is only about 3.7% which implies that the sleep cut-off transistors caused most of the active area penalty. The size of the sleep transistors can be reduced by sharing them as they do in many MTCMOS based designs. This would not only save area but also reduce delays. Hence, we consider the active area *excluding* the sleep-cut off transistors (columns 6 and 7 of Table 3) to be a more meaningful measure of the area penalty. Another important point to note is that, the area overhead reported is only the active area overhead. The effective area overhead is expected to be much smaller once the circuits are placed and routed.

We also estimated the dynamic power consumption associated with using our approach. Intuitively, the dynamic power overhead is expected to be proportional to the active area overhead excluding the sleep transistors (3.7%). However, some of this active area is devoted to the sleep bypass transistors which contribute only their diffusion capacitance to the total switched capacitance during circuit operation. Based on this we estimated the total switched capacitance overhead which is proportional to the dynamic power consumption overhead. The switched capacitance overhead is shown in column 8 of Table 3. The average switched capacitance overhead is only about 1.5% which is also roughly the dynamic power consumption penalty.

The tables 2 and 3 prove the effectiveness of our methodology. Note, that the modified circuits have a lower leakage with *no delay penalty* (or in some cases a delay improvement) and a modest increase in dynamic power consumption. This is an improvement over previous approaches [1, 2, 8, 4] that got similar leakage improvements only at the expense of a delay increase.

6 Conclusions

In this paper we presented an algorithm that replaced gates in a circuit in an effort to reduce the standby leakage of

Ckt.	Original Area(μ^2)	Total New Area(μ^2)	Total New Area Ovh(%)	Sleep Transistor Area(μ^2)	New Area excluding sleep cut-off transistors(μ^2)	Area overhead excluding sleep cut-off transistors(%)	Switched Cap Ovh.(%)
alu2	78.52	96.20	22.52	14.08	82.12	4.58	2.42
alu4	155.42	187.94	20.92	24.87	163.07	4.92	2.68
apex6	157.36	197.15	25.29	34.71	162.44	3.23	1.06
apex7	49.04	66.32	35.24	15.05	51.27	4.55	1.38
C1355	108.20	133.74	23.60	22.34	111.40	2.96	1.38
C432	37.92	46.01	21.33	7.29	38.72	2.11	0.42
C880	83.94	107.56	28.14	20.52	87.04	3.69	1.31
C1908	104.21	134.74	29.30	26.95	107.79	3.44	1.04
C3540	246.42	305.13	23.83	48.84	256.29	4.01	1.69
C6288	672.99	970.35	44.18	260.06	710.29	5.54	1.53
dalu	211.55	259.04	22.45	38.50	220.54	4.25	1.53
des	812.09	1054.80	29.89	209.27	845.53	4.12	1.64
i10	490.08	621.40	26.80	109.84	511.56	4.38	1.79
i1	11.90	13.99	17.56	1.85	12.14	2.02	0.40
i2	50.84	53.99	6.20	2.81	51.18	0.67	0.13
i3	32.28	40.36	25.03	5.00	35.36	9.54	6.37
i6	109.22	124.21	13.72	13.49	110.72	1.37	0.27
i7	147.63	170.96	15.80	21.11	149.85	1.50	0.30
i8	234.59	273.09	16.41	32.37	240.72	2.61	0.75
i9	151.56	179.53	18.45	24.13	155.40	2.53	0.73
t481	166.08	213.81	28.74	40.15	173.66	4.56	2.05
too_large	62.51	80.85	29.34	15.40	65.45	4.70	2.17
Avg			23.85			3.69	1.50

Table 3. Area (Active area) Cost of Using Our Approach

the circuit. The replacement does not necessarily reduce the leakage of the gate being replaced, but helps set the gates in the transitive fanout to their low leakage states. The algorithm involves traversing the circuit from the PIs to the POs, replacing gates as required to try and get as many gates as possible to their low leakage state. We get an average decrease in leakage of about 29% with an active area penalty of about 24%. This leakage decrease is the decrease over the leakage obtained through input vector control alone.

References

[1] A. Abdollahi, F. Fallah, and P. Massoud. Runtime mechanisms for leakage current reduction in CMOS VLSI circuits. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pages 213–218, 2002.

[2] A. Abdollahi, F. Fallah, and M. Pedram. Leakage current reduction in CMOS VLSI circuits by input vector control. *IEEE Trans. VLSI Syst.*, 12(2):140–154, 2004.

[3] 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*, pages 201–204, Jun 2000. http://www-device.eecs.berkeley.edu/ ptm.

[4] L. Cheng, L. Deng, D. Chen, and M. D. F. Wong. A fast simultaneous input vector generation and gate replacement algorithm for leakage power reduction. In *DAC*, pages 117–120, 2006.

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

[6] L. Nagel. Spice: A computer program to simulate computer circuits. In *University of California, Berkeley UCB/ERL Memo M520*, May 1995.

[7] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.

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