Standby Power Optimization via Transistor Sizing and Dual Threshold Voltage Assignment *

Mahesh Ketkar and Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455, USA

ABSTRACT

This paper presents a novel enumerative approach, with provable and efficient pruning techniques, for dual threshold voltage (V_t) assignment at the transistor level. Since the use of low V_t may entail a substantial increase in leakage power, we formulate the problem as one of combined optimization for leakage-delay tradeoffs under V_t optimization and sizing. Based on an analysis of the effects of these two transforms on the delay and leakage, we justify a two-step procedure for performing this optimization. Results are presented on the ISCAS85 benchmark suite favorably comparing our approach with an existing sensitivity-based optimizer.

1. INTRODUCTION

In deep submicron technologies where V_t does not scale with supply voltage, a strategy to improve circuit timing has taken shape in the form of dual V_t technology. The choice of V_t involves a tradeoff since the high V_t leads to higher gate delay and lower standby leakage power dissipation while the low V_t leads to faster gates but with drastically higher standby leakage power dissipation.

Static power-delay tradeoff optimization by dual V_t assignment has been tackled previously in [1–3]. An important drawback in [2, 3] is that V_t assignment is performed at the gate-level, i.e., all of the transistors in a gate are forced to have same V_t, as opposed to considering each transistor separately for assignment, and this eliminates a significant number of good circuit configurations. This acts as our motivation for presenting an algorithm that works at the transistor-level. In practice, process constraints may require the assignment of the same V_t to a group of transistors, and our method is general enough to handle such additional constraints. Our V_t assignment approach is based on explicit enumeration of all the design solutions, made efficient by pruning techniques.

The gate delay as well as the leakage current depend on the V_t and the transistor size, and hence optimizing V_t without considering transistor sizing leaves a significant part of the design space unexplored, and this forms a major drawback of [1–3]. The problem of simultaneous optimization was first introduced in [4], which used a simplistic binary search technique. The work in [5] considerably enhances prior work in removing several of the limitations; however, it employs a sensitivity-based engine to update transistor sizes as well as V_t's, and this may result in solutions far from optimal, especially in the presence of stringent constraints.

* This work was supported in part by a gift from Intel Corporation, by the NSF (CCR-0098117) and by the SRC (99-TJ-714). We would like to thank Dr. Patra from Intel.

We examine the circuit optimization problem of minimizing standby leakage power under area and timing constraints. During the design process, a circuit is expected to meet a set of delay constraints and to occupy an area that is no more than that allotted to it during floorplanning. It can be shown upon proper formulation that the above problem is a mixed integer nonlinear program (MINLP), and our experiments showed that the use of an accurate and exact MINLP solver [6] required large CPU times even on small circuits. Therefore, a computationally efficient, yet accurate, heuristic is essential. The variation in the dependence of the transistor drain current on V_t in the ohmic and subthreshold regions and the nature of a typical area-delay tradeoff curve are used to draw a set of conclusions that form the theoretical underpinnings of our two-step approach, which involves transistor sizing as first step, and our V_t assignment approach as the second step. Since our V_t assignment approach works at transistor level, it fits in well with the transistor sizing, and thus leads to an efficient exploration of the design space at transistor level.

2. LEAKAGE AND DELAY ESTIMATION

The effect of V_t on leakage current can be seen from the following BSIM equations for a single transistor:

$$\begin{aligned} I_{leak} &= I_o e^{(V_{gs}-V_t)/nV_{therm}}(1-e^{-V_{ds}/V_{therm}}), \\ \text{where} \quad I_o &= \mu_0 C_{ox}(W/L){V_{therm}}^2 e^{1.8} \end{aligned} \tag{1}$$

Here μ_0 is the zero bias electron mobility, n is the subthreshold slope coefficient, V_{gs} and V_{ds} are the gate-to-source voltage and the drain-to-source voltage, respectively, V_{therm} is the thermal voltage, and W and L are the transistor width and length, respectively. In practice, for easy estimation from SPICE parameters we fit Equation (1) as a function of transistor V_t and transistor width.

The evaluation of the leakage current of a gate can be made efficient by using the concept of dominant states [5]. Stated in plain English, dominant leakage states, which account for 95% of leakage, for simple gates refer to the leakage states with only one transistor off in the pull-up or pull-down chain. This observation allows us to use the single transistor BSIM leakage current model for leakage estimation, since for a dominant state, each stack contains only one transistor in the subthreshold region. To estimate leakage for optimization purposes, the leakage model of Equation (1) can be multiplied with the probability that the transistor is leaking and the input state is dominant.

The gate delay models used in this work are adopted from [7], and a similar precharacterization procedure is employed. In addition to being accurate in DSM technologies, the model also possesses convexity properties for a fixed

value of V_t, and it is desirable to preserve these so that they can be exploited by the area-delay tradeoff optimization, to arrive at a good solution. However, when V_t is varied, even under a simple model for inverter delay [8], it is easily verified that the delay is not a convex function of the transistor V_t (even assuming that V_t could be changed continuously). Therefore, we do not use V_t as a characterization variable in our convex delay model, but instead, develop multiple pin-to-pin models for each switching scenario, where each model corresponds to a different combination of V_t's of transistors on the resistive path. To avoid developing large number of models for the dual V_t case, we have developed accurate equations to collapse two transistors of different V_t's into only one "equivalent" transistor of either high or low V_t. We validated this collapsing procedure on an extensive of data, and delay values were seen to be within 6% of the actual delay. Our experiments show that the effect of the V_t of a transistor on the switching delay of the previous stage, which is primarily manifested as a change in the loading capacitance, is extremely small, and can safely be neglected. Moreover, in DSM technologies this change is nearly overshadowed by the interconnect capacitance. On a 0.1μm technology [9], the change was found to be under 2% in most cases.

3. DUAL V_T ASSIGNMENT

A V_t assignment algorithm is presented in this section, and is subsequently utilized in an unified approach for leakage power minimization, presented in the next section.

3.1 Algorithm

A combinational circuit is represented as a directed acyclic graph where each node corresponds to a gate and each edge corresponds to an interconnection between gates. The graph is decomposed into fanout-free regions (as in technology mapping algorithms such as [10]), and the enumerative techniques are used to select V_t's within each such region. We assign a level to each gate using the topological sort algorithm; the primary inputs correspond to level 0, while a gate is at level i if the maximum level among all its inputs is $i-1$. The enumeration, which starts from the primary inputs and proceeds in the order of increasing level, processes every gate for its all possible V_t-states, and calculates the corresponding arrival times and leakage.

If we represent a high V_t as 0 and a low V_t as 1, then the V_t assignment, X, of a gate, can be encapsulated as a binary number whose q^{th} bit corresponds to the V_t of the q^{th} transistor in the gate. A gate is then defined to be in a V_t-state k if k is the decimal equivalent of the binary number, X_j, representing the V_t settings of the gate j. At any particular gate, the enumeration stores all solutions that are not suboptimal. This is done by maintaining a set of tuples of the form $\{X, L, AR, AF, IS\}$, where X is the V_t-state of the gate under consideration, L is the total leakage of the tree rooted at the node representing current gate, IS represents the set of input tuples (one tuple corresponding to each input), while AR and AF are the rise and fall arrival time at the output. Our accurate gate delay model also uses the input transition time as a variable, and hence we include those in the tuples as well; however, these are omitted from the discussion for simplicity. The routine get_new_tuple generates the output tuple set by setting the arrival times at the gate inputs to those corresponding to the input tuples, and then calculating the arrival times, AR and AF, at the gate output and the corresponding leakage under allowable V_t-settings for the gate. If a generated tuple is not provably suboptimal, it is inserted in the tuple set of the gate j.

```
1. Algorithm : assign_vt
2.     for each level i
3.         for each gate j in level i
4.             for each V_t-state k
5.                 for each set of tuple combinations
                   at the input nodes of j
6.                     get_new_tuple();
7.                 prune_tuple_set();
8.                 if multifanout gate
9.                     process_tree();
10. Algorithm : get_new_tuple
11.        AR_j = AF_j = L_j = 0;
12.        L = leakage(j);
13.        for each input k of gate j
14.            evaluate AR_kj = AF_k + DR_kj;
15.            evaluate AF_kj = AR_k + DF_kj;
16.            L_j = L_j + L_k;
17.            check for suboptimality;
18.            if provably suboptimal then continue;
19.        insert current tuple in the set of tuples,
           tuple_set(j)
20. Algorithm : prune_tuple_set
21.     for each tuple j in the set
22.        for each tuple i|i > j, in the set of tuples
23.          if AR_i ≥ PF × AR_j and AF_i ≥ PF × AF_j
24.            if L_i ≥ PF × L_j then remove
25.            tuple i from the set;
26.          elseif AR_j ≥ PF × AR_i and AF_j ≥ PF × AF_i
27.              if L_j ≥ PF × L_i then remove
28.              tuple j from the set;
29.              proceed to next tuple;
30. Algorithm : process_tree
31.     for each tuple at the node
32.         if AR_j < RR_j and AF_j < RF_j
33.           if L < L(mintuple)
34.              mintuple = current tuple;
35.     assign X(mintuple) to current gate;
36.     input_queue insert IS of mintuple;
37.     while input_queue is not empty
38.         current_tuple = pop input_queue;
39.         current_gate = gate of current_tuple;
40.         if V_t assigned to current_gate, next;
41.         assign X(current tuple) to current_gate;
42.         if IS not empty insert IS in input_queue;
```

Figure 1: Pseudocode for V_t assignment

At every multiple fanout node, an assignment of V_t is chosen for the tree rooted at that node based on the arrival time constraint. At root node, the tuple that has the lowest leakage among all of the tuples that meet the arrival time constraints at that node is selected. If none of the tuples can meet the constraints, the tuple that has minimum offset from the required arrival time is selected. Based on the ID's of the tuples at each of its input nodes, the inputs are assigned the corresponding V_t settings, and the backward traversal is performed until the tree has been exhausted.

Figure 2 shows an example of the enumeration. Consider a sample tuple, shown in bold face, at node j, which corresponds to an inverter, where the number outside the brace represents the ID of the tuple. The first value in the tuple is 2 which means that the V_t-state is 2, corresponding to a V_t assignment of {1,0}, for the transistors of the gate j. The second number represents the total local leakage, i.e., the leakage of node j plus the leakage corresponding to its transitive fanins, the third and fourth values are AF and AR, respectively, and the last field is a vector that holds the ID of

the input tuple. When this tuple propagation reaches node m, which is a multifanout node, the best tuple that meets the timing constraints is selected and the corresponding V_t is assigned to m. The algorithm then checks the ID's of input tuples, $< 10, 1 >$, for the left and right child, respectively, and assigns V_t's corresponding to tuple 10 to node l and corresponding to tuple 1 to node j, and so on.

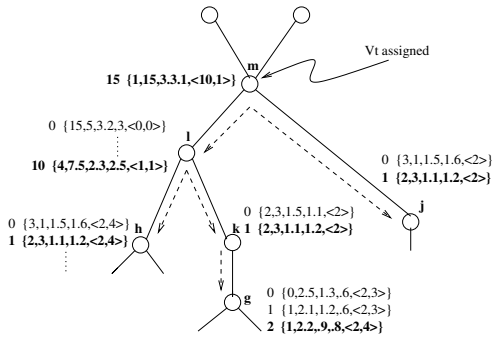

Figure 2: Enumeration for V_t assignment

3.2 Pruning techniques

Since the algorithm employs explicit enumeration, use of pruning techniques is necessary to make it efficient. Due to the dependence of the gate delay and transition time on the fanout loading, there is no straightforward optimal substructure property enabling use of dynamic programming.

Provably suboptimal tuples

A provably suboptimal tuple can be a tuple that satisfies one of the following three properties:

(1) There exists another tuple whose arrival times and leakage current are all less than those in the current tuple.

(2) The rise (fall) time for the tuple, when it is added to the low-V_t delay from current gate to the root of the fanout-free tree, results in a delay that exceeds the required rise (fall) time at that multi-fanout gate.

(3) The tuple has rise and fall times such that when they are added to the high-V_t delay from the output of the gate under consideration to the root of the tree, the resulting delays both satisfy the required times at the multifanout gate, but with a higher leakage than a previously computed tuple that also satisfies those required times.

Quasi-suboptimal tuples

More aggressive pruning techniques are required to control the number of tuples, especially when the fanout-free regions are large. For this purpose, prune_tuple_set removes quasi-suboptimal tuples. A tuple is quasi-suboptimal if there exists another tuple such that multiplication of the arrival times and leakage of that tuple by a small prune factor makes the current tuple provably suboptimal. In practice, different pruning factors can be used for arrival times and leakage.

It is important to explain why the use of our pruning techniques lead to the removal of a significant number of tuples. While the number of tuples can theoretically increase exponentially, this does not happen in practice.

The first reason for existence of suboptimal tuples is the nature of enumeration technique. The extensive spread of variables such as input arrival times, input transition times, and V_t-settings can produce very close tuples. For example, consider two scenarios for an inverter:

Case 1: The V_t-setting for the inverter is such that the nmos transistor is set to high V_t, and the input tuple has low arrival times and high leakage.

Case 2: The V_t-setting for the inverter is such that the nmos transistor is set to low V_t, and the input tuple has high arrival times and low leakage.

In the first case, the inverter delay is high but the input arrival times and the inverter leakage are low, and in the second case inverter delay is low, due to low V_t of switching nmos, but the arrival times and input leakage are low. Hence the two tuples produced at the output can lie in close proximity. In case of gates with higher number of inputs several combinations of arrival times and V_t-settings can lead to generation of very close tuples.

Another reason for tuple pruning is the phenomenon of *input dominance*. The concept can be explained using a noninverting two-input gate, C, shown in Figure 3. The rise and fall times in Tuple 1 at input A of the gate are both higher than those in all of the tuples at input B. Assuming a unit pin-to-pin delay from both input pins, it can be seen that the input tuple combinations will create three tuples at the output with the same arrival times, which are all dictated by tuple at the input A. We say that the tuple 1 at input A *dominates* the tuples at input B. Out of the three tuples at gate C, the highlighted tuple is the only nonsuboptimal tuple, since it has the least leakage among three tuples and only that tuple will be preserved, and the struck out tuples will be pruned

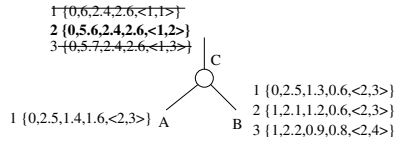

Figure 3: Input dominance

4. UNIFIED ALGORITHM

We propose a heuristic that separates the optimization into two distinct steps: transistor sizing and dual V_t assignment. The key issue is to decide whether to carry out sizing prior to V_t assignment, or vice versa, and to analyze whether such a separation can be logically justified. Let us consider the first order equation for inverter delay [8], given below:

$$t_f = \frac{2C(V_t - 0.1V_{DD})}{\beta_n (V_{DD} - V_t)^2} + \frac{2C}{\beta_n V_{DD}(1-r)}\left[\frac{r-0.1}{1-r} + \frac{1}{2}ln(19-20r)\right] \quad (2)$$

where r is V_t/V_{DD}. Although Equation (2) may be numerically inaccurate in DSM technologies, it has a good fidelity with accurate models. From Equations (1) and (2), we notice the following trends:

Observation 1: The leakage current varies exponentially with V_t, but linearly with the transistor width.

Observation 2: Delay decreases linearly with increasing width, and super-quadraticaly with decreasing V_t.

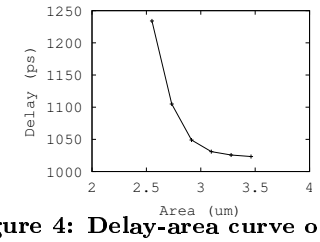

Figure 4: Delay-area curve of C499

Let us examine a typical delay-area trade-off curve for applying only transistor sizing on a combinational circuit; an example is shown in Figure 4. The curve shows a distinctive knee region, beyond which the improvement in the delay per unit increase in the area decreases significantly. This leads

us to the next observation:
Observation 3: Up to the knee point, sizing results in a linear increase in the area, with a very small rate of increase. Beyond this region, the area increases exponentially. Since the leakage current varies linearly with the transistor width, it shows the same trends as the area.

From these three observations, we can conclude that:
(1) Up to the knee of the curve, both sizing and V_t optimization cause the delay to reduce, but from Observation 3, their effects on the leakage power are different: while V_t alteration causes an exponential jump, a sizing step causes a much more sedate and linear change.
(2) Beyond the knee of the curve, both sizing and V_t alteration show an exponential increase in the leakage power, while the latter is more effective at delay reduction.

Therefore, it makes engineering sense to use a two-stage optimization as follows:
Step 1: Reduce delay by sizing until the knee of the area-delay curve is reached and hence obtain linear increase in the leakage current. We set all transistors to high V_t and formulate the transistor sizing problem as one of delay optimization under area constraints. We use a sensitivity-based heuristic similar to TILOS [11] for optimization in this step.
Step 2: Beyond the knee point, since the leakage current increases exponentially regardless of whether the sizing is used or the V_t assignment is used, V_t assignment alone can be used since it is more effective in delay reduction, while maintaining constant area. Hence, we perform dual V_t assignment, for the sizes obtained in the first step. Based on the arrival times obtained after sizing and the required times at the output, rise and fall required times are assigned at the output of every multifanout gate, using CPM.

5. RESULTS

The algorithms are implemented as a CAD tool called SATVA (**S**izing **A**nd **T**hreshold **V**oltage **A**ssignment), and the technology parameters provided by a 0.1μm model [9] are used in the experimentation. The probability values are obtained using a probability propagation routine, where all of the inputs are assigned a probability, $P_1(i) = 0.5$, where $P_1(i)$ represents the probability that the signal i is at logic 1. Before application of Step 2, delays from every gate to root of the fanout free tree to which it belongs are calculated and stored, for both all high-V_t and all low-V_t settings, and are used in provable pruning. The experiments are carried out on a Sun Ultra 10. ISCAS85 benchmark circuits, mapped to simple gates, are used as inputs to our optimization algorithm.

For the purpose of evaluating the efficacy of V_t assignment algorithm under constant transistor sizes, we implemented a priority-based backtracking (PB) approach similar to [1], and both algorithms are run at transistor-level. Our algorithm resulted in 4x improvement in leakage on average compared to PB.

Table 1 provides results of optimization by our tool and by a sensitivity-based algorithm similar to [5]. The first column lists the circuit name and the target area, and the second and third columns provide the original and the target delays, respectively. We use the circuit configuration obtained after the sizing step of our tool as the input to SBA. The remaining columns list the leakage current and CPU times for the two methods. It can be seen that our tool delivers a large performance improvement over SBA.

Circuit	Delay (ps)		Leakage (nA)		CPU (s)	
A (μm)	Orig	Target	SBA	SATVA	SBA	SATVA
C1908	2240	1570	923	969	519	192
391		1460	1025	1110	568	440
		1340	1365	1411	824	175
C2670	2615	1830	1374	1374	407	94
564		1700	1482	1516	560	96
		1570	1795	1700	879	114
C3540	3155	2210	1965	1990	1167	196
788		2050	2336	2218	1726	895
		1890	2905	2600	5556	1945
C5315	2786	1950	2669	2658	1233	317
1163		1810	2856	2769	1714	317
		1670	3365	2918	4900	322
C6288	7760	5430	5275	3831	5400	992
1080		5040	6657	4583	9400	991
		4650	8903	5392	9436	993
C7552	2280	1600	4197	4112	10600	6378
1646		1480	4891	4498	11700	6406
		1370	6632	5129	11125	6448

Table 1: Comparison of SBA [5] and SATVA.

From the results, we can see that our algorithm performs approximately the same as SBA on loose constraints, but the quality of the results improves, as expected, when the constraints are tightened, it performs little worse as compared to SBA on smaller circuits, but vast improvements are obtained for large circuits. For example, the average improvement for C6288 and C7552, is 21%.

We also tested our algorithm when a set of transistors is constrained to have the same V_t, we performed stack-level (all transistors in a stack have same V_t) and gate-level assignments. This requires that enumeration need consider only those V_t-states where the stack-level and the gate-level assignment constraints are satisfied, respectively. Our results showed that assignment at transistor level results in 3.5% improvement on average in leakage compared to stack-level, and 32% compared to gate-level, underscoring the relevance of an approach that can work at these finer levels.

6. REFERENCES

[1] L. Wei *et al.*, "Mixed-Vth (MVT) CMOS circuit design methodology for low power applications," *Proc. DAC*, pp. 430–435, 1999.

[2] Q. Wang and S. B. K. Vrudhula, "Static power optimization of deep submicron CMOS circuits for dual V_t technology," *Proc. ICCAD*, pp. 490–496, 1998.

[3] V. Sundararajan and K. Parhi, "Low power synthesis of dual threshold voltage CMOS VLSI circuits," *Proc. ISLPED*, pp. 139–144, 1999.

[4] P. Pant, V. De, and A. Chatterjee, "Simultaneous power supply, threshold voltage, and transistor size optimization for low-power operation of CMOS circuits," *IEEE Trans. on VLSI Systems*, vol. 6, pp. 538–545, Dec 1998.

[5] S. Sirichotiyakul *et al.*, "Stand-by power minimization through simultaneous threshold voltage selection and ciruit sizing," *Proc. DAC*, pp. 436–441, 1999.

[6] University of Dundee, *User Manual for MINLP_BB*, 1999.

[7] M. Ketkar *et al.*, "Convex delay models for transistor sizing," *Proc. DAC*, pp. 655–660, 2000.

[8] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Reading, MA: Addison-Wesley, 2nd ed., 1994.

[9] "Predictive technology model." http://www-device.eecs.berkeley.edu/~ptm/.

[10] K. Keutzer, "DAGON: Technology mapping and local optimization," *Proc. DAC*, pp. 341–347, 1987.

[11] J. P. Fishburn and A. E. Dunlop, "TILOS : A posynomial programming approach to transistor sizing," *Proc. ICCAD*, pp. 326–328, Nov. 1985.