

Runtime Mechanisms for Leakage Current Reduction in CMOS VLSI Circuits^{1, 2}

Afshin Abdollahi
University of Southern California
(310) 592-3886
afshin@usc.edu

Farzan Fallah
Fujitsu Laboratories of America
(408) 530-4544
farzan@fla.fujitsu.com

Massoud Pedram
University of Southern California
(213) 740-4458
pedram@ceng.usc.edu

Abstract. This paper describes two runtime mechanisms for reducing the leakage current of a CMOS circuit. In both cases, it is assumed that the system or environment produces a "sleep" signal that can be used to indicate that the circuit is in a standby mode. In the first method, the "sleep" signal is used to shift in a new set of external inputs and pre-selected internal signals into the circuit with the goal of setting the logic values of all of the internal signals so as to minimize the total leakage current in the circuit. This minimization is possible because the leakage current of a CMOS gate is a strong function of the input combination applied to its inputs. In the second method, NMOS and PMOS transistors are added to some of the gates in the circuit to increase the controllability of the internal signals of the circuit and decrease the leakage current of the gates using the "stack effect". This is, however, done carefully so that the minimum leakage is achieved subject to a delay constraint for all input-output paths in the circuit. In both cases, Boolean satisfiability is used to formulate the problems, which are subsequently solved by employing a highly efficient SAT solver. Experimental results on the circuits in the MCNC91 benchmark suite demonstrate that it is possible to reduce the leakage current by up to 70% in VLSI circuits at the expense of a very small overhead.

Categories and Subject Descriptors:

B.7.1. [Integrated Circuits]: Types and Design Styles, VLSI

General Terms: Algorithms and Design

1 Introduction

The rapid increase in the number of transistors on chips has enabled a dramatic increase in the performance of computing systems. However, the performance improvement has been accompanied by an increase in power dissipation; thus, requiring more expensive packaging and cooling technology. Historically, the primary contributor to power dissipation in CMOS circuits has been the charging and discharging of load capacitances, often referred to as the dynamic power dissipation. This component of power dissipation is quadratically proportional to the supply voltage level. Therefore, in the past, chip designers have relied on scaling down the supply voltage to reduce the dynamic power dissipation. Maintaining the transistor switching speeds requires a proportionate downscaling of the transistor threshold voltages in lock step with the supply voltage reduction. However, threshold voltage scaling results in a significant amount of leakage power dissipation due to an exponential increase in the sub-threshold leakage current conduction. Borkar in [2] predicts a 7.5 fold increase in the leakage current and a five-fold increase in total energy dissipation for every new microprocessor chip generation. Unlike the dynamic power, which depends on the average number of switching transistors per

clock cycle, the leakage power depends on the number of on-chip transistors, regardless of their average switching activity. The input pattern dependence of the leakage current makes the problem of determining the leakage power dissipated by a circuit a difficult one. This statement is true even when runtime statistics about the active versus idle times for a circuit are known. This is because by applying the minimum-leakage producing input combination to the circuit when it is in the idle mode, we can significantly reduce the leakage power dissipation of the circuit. Consequently, identification of a Minimum Leakage Vector (MLV) is an important problem in low power design of VLSI circuits.

In this paper several runtime mechanisms for leakage current reduction of CMOS VLSI are introduced. Our methods find the MLV of a circuit and the optimum way of modifying the circuit to reduce its leakage current using a Boolean satisfiability formulation. In the next section a review of a number of the leakage reduction techniques is presented. In Section 3, we describe a method for finding the MLV and its corresponding leakage current. Our method is based on constructing a Boolean network for computing the leakage current of a VLSI circuit and solving a series of Boolean satisfiability problems corresponding to that network. We use an incremental satisfiability solver technique to speedup the operation [14]. We minimize the leakage current by using an MLV to drive the circuit while in the standby mode. In Section 4, two improved mechanisms for leakage current reduction are introduced. The basic idea is to increase the controllability of the internal signals of a circuit. Using multiplexers or modifying the internal gates of the circuit achieves this. Experimental results and conclusion are presented in Sections 5 and 6.

2 Previous work

In this section, we briefly review a number of commonly used leakage reduction techniques.

2.1 Leakage Reduction by Input Vector Control

Many researchers have used models and algorithms to estimate the nominal leakage current of a circuit [3][4]. The minimum and maximum leakage currents of a circuit have been estimated using a greedy heuristic in [5]. Because of the transistor stacking effect, the leakage of a circuit depends on its input combination [5]. As the operational state of the transistors that constitute a CMOS gate are determined by their input signal values, the goal can be expressed as finding the input pattern that maximizes the number of disabled ("off") transistors in all stacks across the circuit [6]. The authors in [7] provided an estimation of the maximum leakage current by greedily assigning input combinations of logic blocks that result in high leakage currents. All the above methods can be used to determine the minimum-leakage vector and to further exploit the stacking effect by inserting transistors in the leaky sections of a circuit [8]. Another possibility is to perform an exhaustive circuit-level simulation for all input patterns to find the pattern with the minimum leakage current. However, this approach is not practical for large circuits. In [9], the authors used probabilistic methods to reduce the number of simulations necessary to find a solution with a desired accuracy. Having found the minimum leakage pattern, one can use this vector to drive the circuit while in standby mode. This requires the addition of a number of multiplexers at the primary inputs of the circuit. The multiplexers are controlled using a sleep signal. Because the power reduction using this technique can be achieved only for long sleep periods, a threshold is used to activate the sleep signal only if the sleep period is long enough.

¹This research was sponsored in part by DARPA PAC/C program under contract no. DAAB07-00-C-L516.

²Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

Copyright 2002 ACM 1-58113-475-4/02/0008...\$5.00.

2.2 Leakage Reduction by Increasing the Threshold Voltages

One way of decreasing the leakage current is increasing the threshold voltages of transistors. There are several ways to do this, but in all of them some process technology modification is necessary. However, this may not be always possible. Another approach is to use high-threshold voltage devices on non-critical paths so as to reduce the leakage power while using low-threshold devices on critical paths so that the circuit performance is maintained. This technique requires an algorithm that searches for the gates where the high-threshold voltage devices can be used [11]. This technique has been called the Dual V_{th} CMOS. In Dynamic Threshold MOS (DTMOS), the body and the gate of each transistor are tied together such that when the device is off, the leakage is low. If the device is on, then the current will be high [13]. Among the techniques that dynamically modify the threshold voltage during runtime, the classic example is Standby Power Reduction (SPR) or Variable Threshold CMOS (VTCMOS). In this method V_{th} is raised during the standby mode by making the substrate voltage either higher than V_{dd} (for P transistors) or lower than ground (for N transistors). However, this technique requires an additional power supply, which may not be attractive in some commercial designs. A technique presented in [12] successfully solves this problem and applies the technique to a commercial digital signal processor. The architectural support needed to use VTCMOS can be done in hardware or software. There is a large performance penalty due to the time required removing the substrate voltage to return to the normal operation mode. Noise immunity problems have been reported when the substrate voltage is changed, but since in this case the technique is applied when the system is idle, there is no negative effect on the normal operation of the circuit.

2.3 Leakage Reduction by Gating the Supply Voltage

The last approach considered is power supply gating. There are many ways in which this technique can be implemented, but the basic idea is to shut down the power supply so the idle units do not consume any power. This can be done using some high threshold transistors called sleep transistors [1]. If the threshold voltages of sleep transistors are changed at runtime, the triple-well technology is required. Another possibility is to use Multiple-Threshold Voltage CMOS (MTCMOS) [10]. In MTCMOS, a high threshold device is inserted in series with low threshold transistors creating a sleep transistor. This creates virtual supply and ground rails whose voltage levels are very close to the real supply and ground lines because of the very small on-resistance of the inserted high- V_{th} transistors. In practice, only one virtual rail (usually the virtual ground) is used. Normally, one sleep transistor per gate is used, but larger granularities are possible, which require fewer transistors. The problems with this technique are reduced performance and noise immunity.

3 Leakage Minimization by Input Vector Control

By applying a minimum leakage vector (MLV) to a circuit, it is possible to decrease the leakage current of the circuit when it is in the standby mode. We assume that the environment in which the circuit is placed e.g., with the aid of a power management unit, generates a *SLEEP* signal for the circuit. This signal is then used to initiate the application of the MLV to the circuit inputs. To use this method for leakage reduction, it is necessary to find an input vector that causes the minimum leakage current in a VLSI circuit. A *trivial lower (upper) bound* on the leakage current is the sum of the minimum (maximum) leakage currents of all logic gates in the circuit. However, this may not correspond to any feasible solution because the input combination that produces the minimum (maximum) leakage in some gate, $gate_i$, may conflict with the one that produces the minimum leakage for another gate, $gate_j$. In the remainder of this section, we describe an algorithm for finding an MLV for a given combinational logic circuit. More precisely, given a combinational logic circuit description, we first construct a Boolean network, which computes the total leakage of that circuit. We call the resulting circuit a *Leakage Computing Network (LCN)*.

Next from the LCN description, we write a set of Boolean clauses that capture the leakage current of the original circuit. We employ a SAT solver to find an input vector that results in a leakage less than a given number C . Next, we perform a linear search on the value of C to find the MLV. Finally, we modify the original circuit by adding a number of multiplexers to shift in the MLV when the circuit enters the idle mode. Notice that the LCN is only used as a computational tool and the only actual hardware are the original circuit and the final circuit (which is augmented by the multiplexers and MLV vector). Leakage current of a logic gate depends on its input values. Let $leakage(X_j)$ be the leakage current of the j^{th} gate of a circuit under the immediate input vector combination X_j . Notice that $leakage(X_j)$ can be written as a sum of up to 2^n terms, where n is the number of inputs of the gate. For example, the following equation gives the leakage current for all input values of a two-input *NAND* gate:

$$Leakage(X_j) = \bar{X}_{j1} \bar{X}_{j0} L_{00} + \bar{X}_{j1} X_{j0} L_{01} + X_{j1} \bar{X}_{j0} L_{10} + X_{j1} X_{j0} L_{11}$$

where L_{pq} is the leakage current of the gate when $X_{j1}=p$ and $X_{j0}=q$. Without loss of generality, we multiply all gate leakage values with a large constant number to make them integer values. The leakage current minimization problem can then be stated as follows:

Given circuit-induced logic dependencies among X_j 's, find a primary input vector that minimizes $\sum_j leakage(X_j)$ for all gates in the circuit.

The above cost function can be directly implemented in the LCN by using adders and multiplexers. However, to decrease the number of adders, we use the following approach. First we compute the sum of all cost function terms that correspond to some leakage value L_{kl} . Next we compute sum of the results. As an example, consider a circuit with two *NAND* gates, denoted by $gate_i$ and $gate_j$. In a straightforward LCN realization, the following sum is computed:

$$Leakage(X_i) + Leakage(X_j) = (\bar{X}_{i1} \bar{X}_{i0} L_{00} + \bar{X}_{i1} X_{i0} L_{01} + X_{i1} \bar{X}_{i0} L_{10} + X_{i1} X_{i0} L_{11}) + (\bar{X}_{j1} \bar{X}_{j0} L_{00} + \bar{X}_{j1} X_{j0} L_{01} + X_{j1} \bar{X}_{j0} L_{10} + X_{j1} X_{j0} L_{11})$$

where X_i is a Boolean variable and L_{ij} is a fixed-length vector of Boolean variables corresponding to the binary representation of the actual leakage value. The LCN size can be reduced if we rearrange the terms as follows:

$$Leakage(X_i) + Leakage(X_j) = (\bar{X}_{i1} \bar{X}_{i0} + \bar{X}_{j1} \bar{X}_{j0}) L_{00} + (\bar{X}_{i1} X_{i0} + \bar{X}_{j1} X_{j0}) L_{01} + (X_{i1} \bar{X}_{i0} + X_{j1} \bar{X}_{j0}) L_{10} + (X_{i1} X_{i0} + X_{j1} X_{j0}) L_{11}$$

The reason is that in the latter case, for each leakage value, instead of computing the sum of n terms each with m bits, we compute the sum of n single-bit numbers and then multiply the result with an m -bit number. The first approach needs $m(n-1)$ single-bit adders, while the second one requires $n-1+m \log n$ single-bit adders. Thus, the second approach is more efficient. To compute the total leakage in our approach, we use a decoder for each gate. As an example consider a 2-input gate with 4 different leakage values corresponding to 4 different combinations of its inputs. Figure 1 shows a 2-to-4 decoder associated with this gate in the LCN. In this figure, D_{ij}^k values represent the input combination ij of $gate_k$.

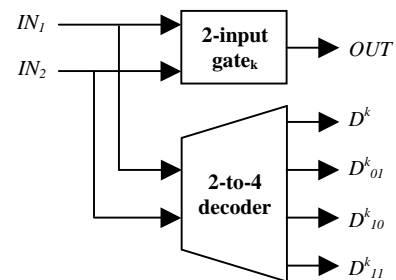


Figure 1. A 2-to-4 decoder indicating input combinations of a 2-input logic gate.

Figure 2 shows the LCN structure for computing the total leakage current of all gates in the original circuit that perform the same Boolean operation (e.g., two-input *NAND*). The one's counters in this figure count the number of D_{ij}^k variables that are assigned a value of *ONE*. For example, if there are 50 two-input *NAND* gates and 20 of them receive input combination 00, while 15, 10 and 5 gates receive 01, 11, and 10 input combinations, respectively, then the total leakage of all two-input *NAND* gates in the circuit will be $20L_{00}+15L_{01}+10L_{11}+5L_{10}$.

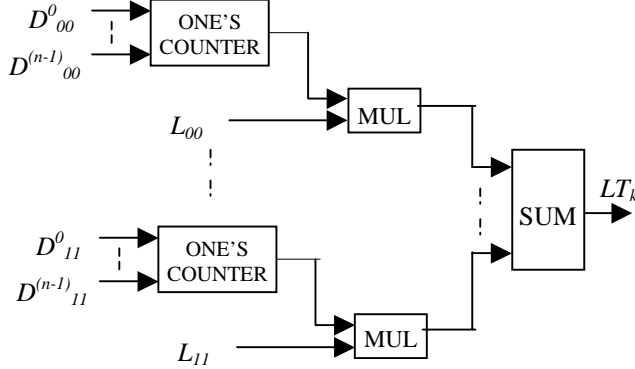


Figure 2. Contribution of all gates of type k to the total leakage.

Notice that when the leakage current of a gate type for a specific input combination is equal to that of another gate for some other input combination, it is possible to share the logic structures between them to improve the size efficiency of the LCN. The total leakage current of the circuit is computed by summing up all LT_k values corresponding to all gate types in the original circuit. Suppose we are interested in finding a vector whose leakage current is less than a given number C . To do this, we compare the total circuit leakage with C . Figure 3 shows the circuit realization for comparing the total circuit leakage with C .

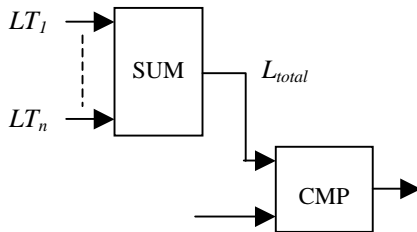


Figure 3. Comparing circuit leakage with C .

We model the circuit in Figure 3 using Boolean clauses as described in [15].

For example if $n=2$, $LT_1=[a_1 a_0]$ and $LT_2=[b_1 b_0]$, then the summation of these two vectors is $L_{total}=[s_2 s_1 s_0]$. The Boolean description of the relation between a_0 , b_0 , and s_0 is $s_0=XOR(a_0, b_0)$ and this Boolean relation can be described by four clauses:

$$\begin{aligned} Clause_1 &= a_0 + b_0 + \bar{s}_0, & Clause_2 &= a_0 + \bar{b}_0 + s_0, \\ Clause_3 &= \bar{a}_0 + b_0 + s_0, & Clause_4 &= \bar{a}_0 + \bar{b}_0 + \bar{s}_0 \end{aligned}$$

Algorithm LIN_SEARCH_FOR_MLV:

1. Find the trivial bounds on leakage current, LB and UB described in the beginning of section 3
2. $C = UB$, $mlv = \{ \}$
3. Write Boolean clauses to model the circuit leakage and the condition that $total_leakage \leq C$
4. Solve the resulting SAT problem
5. If there is no solution, stop; $C + 1$ is the minimum leakage and mlv is the solution
6. $mlv =$ the vector found by the SAT solver
7. $C = C - 1$
8. If $C < LB$, stop; $C + 1$ is the minimum leakage and mlv is the solution
9. Go to step 3

The above algorithm performs a linear search on the values between LB and UB to find the minimum leakage current. The search starts from UB and proceeds toward LB . During the search all problems are feasible except the last one. Note that the constraints corresponding to $total_leakage \leq C - 1$ are tighter than the ones corresponding to $total_leakage \leq C$. Thus, every solution of iteration $i+1$ is a solution of iteration i . In every iteration, the SAT solver produces many conflict clauses during the search for the answer.³ We use this fact to speedup the search by using the conflict clauses that are generated during the i^{th} iteration and adding new clauses to them to model the $(i+1)^{th}$ iteration. This is instrumental in substantially decreasing the computation time.

It is possible to start the search from LB towards UB . In this case all problems except the last one are infeasible. Because this formulation does not permit the reuse of the conflict clauses, it is slower than the one described previously. A binary search, rather than a linear search may also be used. Again we note that a binary search does not permit the reuse of the conflict clauses. Furthermore, the decrease in the number of iterations (sub-problems) tends to be very small compared to the linear search. Therefore, using a linear search algorithm provides the best runtime. After finding the MLV, we use it to drive the circuit every time the *SLEEP* signal is activated. This can be accomplished by using some multiplexers controlled by the *SLEEP* signal to drive the inputs of the circuit. Simplifying the multiplexers based on the fact that one input of each multiplexer is a constant 0 or 1 reduces the hardware cost. Figure 4 shows the input driver for two bits $\{a_1, a_0\}$ assuming the min leakage vector is $\{1, 0\}$.

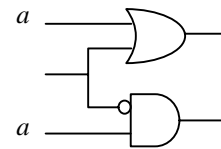


Figure 4. Input driver for min leakage vector $\{1,0\}$

4 Leakage Reduction by Adding Control Points

In the previous section, we reduced the leakage current by using an input vector control mechanism. However, in circuits with large logic depth, an externally applied input vector may effectively control only the gates that are close to primary inputs. If we find a way to directly control at least some of the internal nodes of a circuit, we can further reduce the leakage of the circuit. In this section we introduce two methods to add control points to a circuit to decrease its leakage.

4.1 Using Multiplexers

An easy way to control the value of an internal signal (line) of a circuit is to cut the internal line and insert a 2-to-1 multiplexer that is controlled by the *SLEEP* signal. The two inputs of the multiplexer are the incoming signal and a *ZERO* or *ONE* value decided by the leakage current minimization algorithm. The output is the outgoing signal. Since one input of the multiplexer is a fixed value, instead of the multiplexer, an *AND* gate or an *OR* gate may be used. Figure 5 shows a part of a circuit before and after replacing its internal line by an *AND* gate.

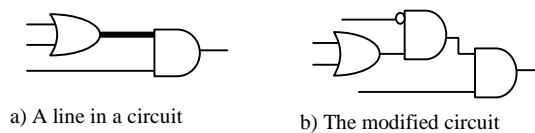


Figure 5. Replacing a line by an *AND* gate.

³ Conflict arises when during the search one or more clauses become unsatisfiable in the current search sub space. The SAT algorithm backtracks from this point and also learns from the conflict by adding one or more conflict clauses to its database. Adding such conflict clauses prevents the algorithm from encountering the same conflict. In other words, clauses prune the search space efficiently [16].

In Figure 5 (b), in the sleep mode, the output of the *AND* gate is *ZERO*; if, based on the result of leakage current minimization algorithm, we need to have a *ONE* on that line in the idled circuit, the *AND* gate has to be replaced by an *OR* gate. The additional *AND* or *OR* gate and the gates in its fanout consume dynamic power when a new value is shifted into the circuit at the beginning and the end of the circuit idle time. This dynamic power consumption is considered to be negligible if the idle time is long enough. We assume that the power management unit for the whole design knows about this overhead and will only activate the *SLEEP* signal if the idle time is expected to be very long. In this paper, we do not concern ourselves with how such a global power management policy for a complete design can be developed and put in place.

When a new control gate is added to the circuit, there will also be additional leakage current associated with that gate. The algorithm that determines the number, type, and insertion location of the control gates inside a combinational logic block must account for the leakage currents of these gates. In the remainder of this subsection we present a method to optimally select a subset of the internal lines in a circuit to be replaced with *AND* or *OR* gates. The method is based on modifying the LCN by adding additional input variables corresponding to the internal lines of the circuit. In other words, for each internal line in the circuit, two new variables *X* and *Y* are introduced. The value of *X* determines whether or not the connection will be replaced by a multiplexer. If $X=1$, then a multiplexer whose inputs are the original line and a variable *Y*, is inserted on that line. The LCN is modified to account for the leakage of the added gate.

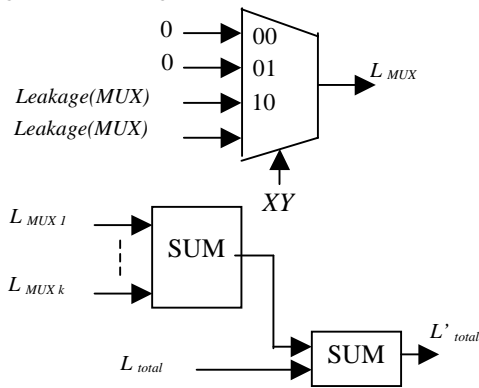


Figure 6. Adding the leakage current of the multiplexer to the total leakage.

Now the problem of minimizing the leakage current can be described as minimizing the value of L'_{total} which is a function of input vector and also variables *X*'s and *Y*'s. By running `LIN_SEARCH_FOR_MLV` on the modified LCN with extra variables (*X*'s and *Y*'s), we can obtain the following:

1. MLV
2. Internal lines on which multiplexers are inserted.
3. *Y* value for each multiplexer and customization of the multiplexer to an *AND* or *OR* gate based on the *Y* value.

Our minimization algorithm finds the optimum subset of internal lines on which multiplexers are inserted. The minimization algorithm considers the advantage of controlling the internal lines in the circuit and weighs it against the disadvantage of additional leakage current due to the required multiplexers. Since the minimization algorithm searches for the minimum leakage solution, if adding any multiplexer helps decrease the leakage, it will be added to the circuit.

4.2 Modifying Gates

The leakage cost of multiplexers serves as a disincentive to employ a large number of these multiplexers in the circuit. In this subsection we propose an alternative method to control the outputs of internal gates in a circuit. Since the new method does not add any gate to the circuit, there is no extra leakage associated with adding a control point to the circuit. Even better, because of the

transistor stack effect, every time we add a control point to the circuit, its total leakage decreases.

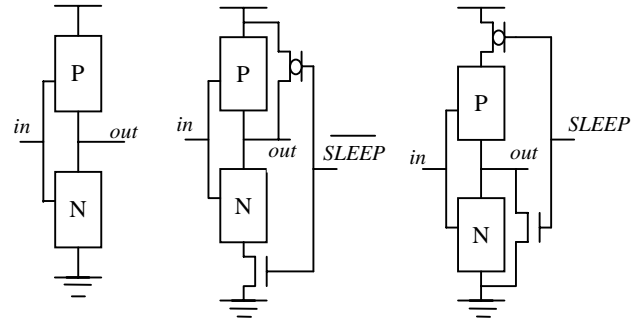
We use two variables *X* and *Y* for each gate in the circuit. The value of *X* determines whether or not a gate in the circuit undergoes some change. The value of *Y* determines the way that the gate is changed. Consider a fully-complementary CMOS gate, $out = g(in)$. Based on the values of *X* and *Y*, which are in turn computed by our leakage minimization algorithm, this gate is changed as follows:

```

If (X==1) out = g(in)
else
  if (Y == 1) out = OR(NOT(SLEEP), g(in))
  else out = AND(SLEEP, g(in))

```

Modifying this gate as described above enables controlling the output of the gate independent of its inputs in the standby mode. In other words, if we must have a *ONE* at the output of the gate when in the standby mode, we replace the gate with $AND(SLEEP, g(in))$. Similarly, if we ought to have a *ZERO*, we replace it with $OR(NOT(SLEEP), g(in))$.



A) $out = g(in)$ B) $out = AND(SLEEP, g(in))$ C) $out = OR(\overline{SLEEP}, g(in))$

Figure 7. A fully-complementary CMOS gate and its two modified circuits.

Figure 7 shows a CMOS gate with its PMOS and NMOS sections and two ways to modify the gate. Note that in both cases a transistor is added in series with one of the N or P sections. This results in a decrease in the leakage of the gate due to the transistor stack effect. The percentage of the reduction depends on the original number of transistors in the gate [8]. Moreover, as mentioned before, this method enables us to control the values of the internal lines in the circuit; thus, reducing the leakage current of the gates in the fanout of the lines. Modifying a gate in this way results in a delay and an area penalty. For example, in case B the high-to-low transition becomes slower, whereas in case C the low-to-high propagation delay is increased. We take the *pin-dependent* propagation delay of a gate to be the average of input-output gate delays for the rising and falling transitions. Obviously, the delay and area penalties depend on the sizes of the added transistors in each case. We size these transistors so that the increase in the delay and the area of each gate is no more than some percentage (Sec. 5.)

In the sequel, we present a method to extend the LCN so that the leakage minimization is performed subject to a delay constraint on all of the primary input to primary output paths in the circuit. The left circuit structure in Figure 8 selects the correct value of the leakage for each gate in the circuit whereas the right structure does the same for delay calculation.

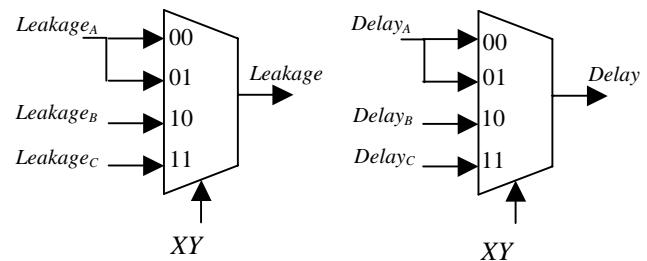


Figure 8. Leakage and Delay current values of a modified gate.

Note that in this figure $leakage_A$ and $delay_A$ denote the leakage current and propagation delay of the gate without modification (i.e., $out=g(in)$). $leakage_B$ and $delay_B$ denote the leakage and propagation delay of the gate modified to $out=AND(SLEEP, g(in))$. $leakage_C$ and $delay_C$ indicate the same for the case where $out=OR(NOT(SLEEP), g(in))$. As in static timing analysis, the gate delay values are used to calculate the maximum delay of the circuit for all input-output paths using the circuit shown in Figure 9. The arrival time of each gate is the maximum of the sum of the arrival time of each of its inputs and the pin-dependent delay from that input to the output of the gate.

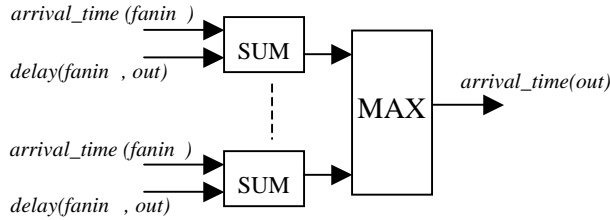


Figure 9. Calculating the output arrival time of a gate.

The maximum delay of the circuit is the maximum of arrival times of its primary outputs. Figure 10 shows the circuit for comparing the maximum delay of the circuit with a given threshold.

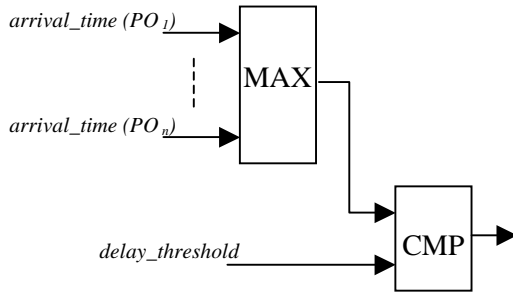


Figure 10. Comparing the maximum delay of the circuit with a delay threshold.

The leakage minimization problem can be stated as that of minimizing the value of L_{total} which is a function of input vector and also variables X 's and Y 's. The leakage minimization has to be performed under the delay constraint illustrated in Figure 10. Therefore, the minimization algorithm should take into account the values of the output of both circuits in Figures 3 and 10 as depicted in Figure 11.

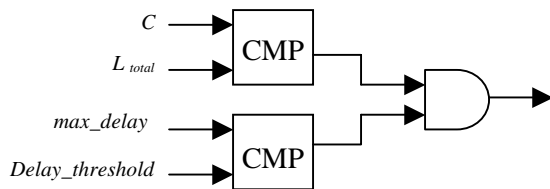


Figure 11. Considering the delay constraint in leakage minimization.

By running LIN_SEARCH_FOR_MLV on the modified LCN with the aforementioned Delay Computing Network (DLN) and variables (X 's and Y 's), we can obtain the following:

1. MLV
2. Gates that are structurally modified.
3. Y value for each modified gate, which identifies the method for modifying the gate.

Our minimization algorithm finds the optimum subset of gates, which are modified. The minimization algorithm considers the advantages of modifying the gates in the circuit (which are controlling internal signal as well as reducing the gate leakage due to the stack effect) and weighs them against the disadvantage of additional delay overhead due to the added transistors.

5 Experimental Results

We applied the proposed mechanisms to reduce the leakage currents of the circuits in MCNC91 benchmark. Each of the circuits was optimized by the SIS script *rugged* and mapped to a technology library using the SIS mapper. We used an industrial library built in 0.18um CMOS technology with a low threshold voltage of 0.2V and a supply voltage level of 1.5V. We used HSPICE simulation to report the leakage current of the gates in the ASIC library for all possible combinations of their inputs. We, therefore, started with a full circuit-level characterization of leakage current of all gates. For each benchmark, we obtained the minimum and the maximum leakage currents and their corresponding input vectors using the method described in Section 3. Figure 12 shows the distribution of the ratio of maximum to minimum for all circuits.

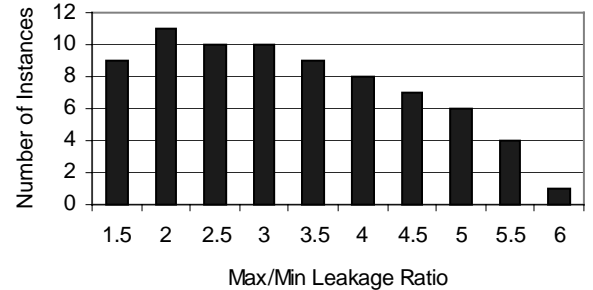


Figure 12. Distribution of maximum over minimum leakage current

Figure 12 depicts our experimental results where we show the max/min leakage distribution for the MCNC91 benchmark suite. The figure, for example, states that 9 of the benchmarks had a max/min leakage ratio between 1.25 (inclusive) and 1.75 (exclusive) whereas 11 had a ratio between 1.75 and 2.25. As it can be seen, the max/min leakage ratio is as high as 6 for some circuits. Therefore, driving the circuit that is placed in the idle mode with a random input vector may result in a significant waste of energy compared to driving the circuit with the MLV. Figure 13 shows the distribution of energy saving achieved by using the input vector control mechanism of Section 3.

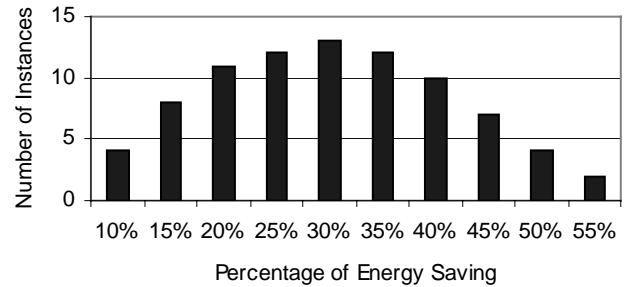


Figure 13. Energy saving of the input vector control mechanism.

Figure 14 shows the distribution of energy saving achieved by using the control point addition mechanism of Section 4.1. As one can see, adding control points to the circuits helps to further reduce the leakage currents.

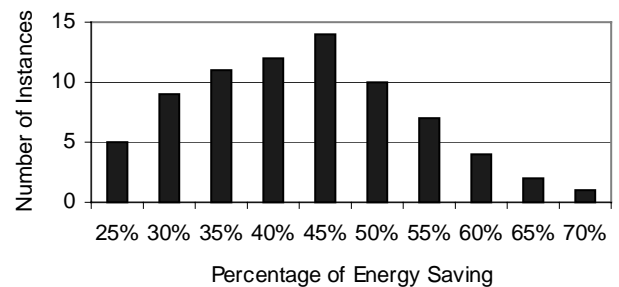


Figure 14. Energy saving for control point addition mechanism

Switching the inputs of a circuit to its MLV and vice versa consumes some dynamic power. The amount of power saved using our runtime leakage control mechanisms depends on the duration of the standby mode for the circuit. For short standby periods, it is not worthwhile to switch between the current input and the MLV. For long standby periods, the energy savings can become quite significant. To make this statement more precise, we calculated the minimum duration of the idle time above which power savings by “shifting” in the MLV becomes possible. Figure 15 shows the distribution of this minimum time (in terms of the number of clock cycles) for MCNC91 benchmark circuits.

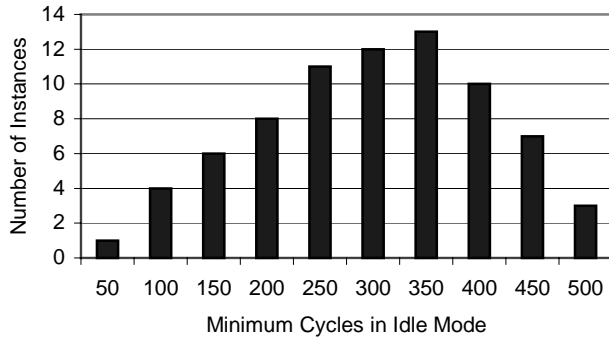


Figure 15. Minimum number of clock cycles that the circuit should stay in the standby mode for the dynamic leakage control to become effective.

The runtime of the algorithm *LIN_SEARCH_FOR_MLV* depends on the number of quantization levels of leakage values. Obviously more quantization levels results in more accuracy and more runtime. Figure 16 shows the distribution of the runtime of the algorithm for 32 and 64 quantization levels.

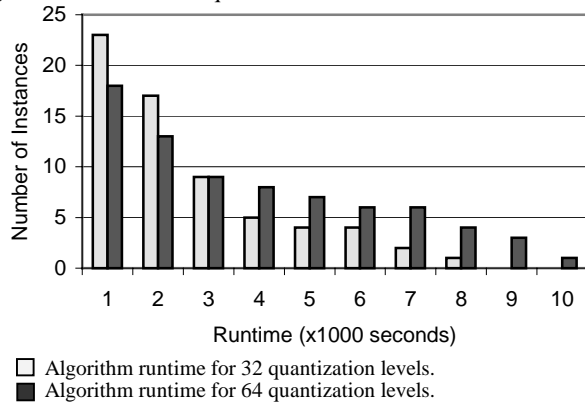


Figure 16 Algorithm runtime for two different quantization levels.

Figure 17 shows the distribution of energy savings for MCNC91 suite that is achieved by using the control point addition mechanism of Section 4.2 under different delay constraints. When we do not allow any speed degradation, only a small number of gates are changed. As a result, the amount of energy saving is on average less than 20% for all the benchmarks. Increasing the limit on the speed degradation helps improve the results as is evident from the figure. For example, with a 15% tolerance on delay, the average energy savings for all the benchmarks is 45-50%. The area overhead is proportional to the number of added transistors and is at most 15%.

6 Conclusion

In this paper we introduced several methods to decrease the leakage current of a circuit. Our methods do not require any modifications in the process technology. Hence, they can be easily used. The experimental results of using our techniques show that up to 70% savings in the leakage current can be achieved at the expense of a small overhead.

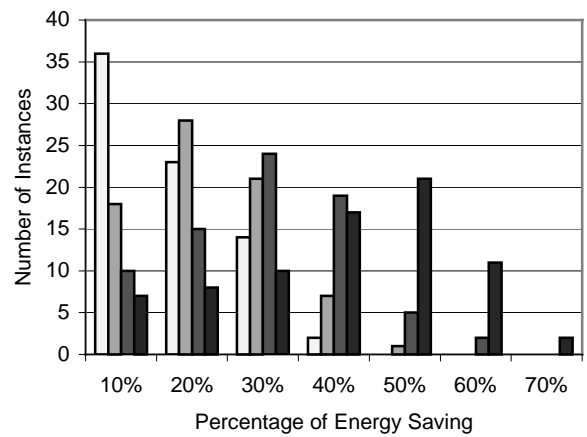


Figure 17 Distribution of energy savings that is achieved by using the control point addition mechanism of Section 4.2 (modifying gates) under different delay constraints.

- Number of Instances for 0% Speed Degradation
- ▒ Number of Instances for 5% Speed Degradation
- ▓ Number of Instances for 10% Speed Degradation
- Number of Instances for 15% Speed Degradation

7 References

- [1] Chandrakasan, A., Bowhill, W. and Fox, F., “Design of High Performance Microprocessor Circuits”. IEEE Press. 2000.
- [2] Borkar, S., “Design Challenges of Technology Scaling”, IEEE MICRO, July-August 1999.
- [3] Ferre, A. and Figueras, J., “Characterization of Leakage Power in CMOS Technologies”, IEEE International Conference on Electronics, Circuits and Systems, Vol. 2, 1998, pp. 85 –188.
- [4] Cheng, Z., Johnson, M., Wei, L. and Roy, K., “Estimation of Standby Leakage Power in CMOS Circuits Considering Accurate Modeling of Transistor Stacks”, ISLPED 98, pp. 239-244.
- [5] Johnson, M., Somasekhar, D. and Roy, K., “Models and Algorithms for Bounds in CMOS Circuits”, IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 18, No. 6, June 1999, pp. 714-725.
- [6] Ye, Y., Borkar, S., and De, V., “A New Technique for Standby Leakage Reduction in High-Performance Circuits,” Symposium on VLSI Circuits, 1998, pp. 40-41.9
- [8] Johnson, M., Somasekhar, D. and Roy, K., “Leakage Control With Efficient Use of Transistor Stacks in Single Threshold CMOS ”, Proceedings of the 36th Design Automation Conference (DAC), June 1999, pp. 442-445.
- [9] Halter J., and Najm, F., “A Gate-level Leakage Power Reduction Method for Ultra Low Power CMOS Circuits, IEEE Custom Integrated Circuits Conference, 1997, pp. 475-478.
- [10] Mutoh, S., Douskei, T., Matsuya, Y., Aoki, T., Shigematsu, S. and Yamada J., “1-V Power Supply High-Speed Digital Circuit Technology with Multi-threshold Voltage CMOS”, IEEE Journal of Solid-state Circuits, pp. 847-854, August 1995.
- [11] Wei, L., Chen, Z., Johnson, M., Roy, K. and De, V., “Design and Optimization of Low Voltage High Performance Dual Threshold CMOS Circuits”, Proceedings of the 35th Design Automation Conference (DAC), 1998, pp. 489-494.
- [12] Kuroda, T., et. al., “A 0.9V 150MHz 10mW 4mm2 2-D discrete cosine transform core processor with variable threshold voltage (VT) scheme,” IEEE Journal of Solid-State Circuits, pp. 1770-1779, November 1996.
- [13] Assaderaghi, F., Sinitsky, D., Parke, S.A., Bokor, J., Ko, P.K. and Hu, C., “Dynamic threshold-voltage MOSFET (DTMOS) for ultra-low voltage VLSI”, IEEE Transactions on Electron Devices, Vol. 44, No. 3, March 1997, pp. 414 –422.
- [14] J. Whitemore, J. Kim, K. A. Sakallah, “SATIRE: A New Incremental Satisfiability Engine,” *Proc. of Design Automation Conference*, 542-545, 2001.
- [15] T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. IEEE Transactions on Computer-Aided Design, 11:4-15, January 1992.
- [16] L. Zhang, C. Madigan, M. Moskewicz, S. Malik, “Efficient Conflict Driven Learning in a Boolean Satisfiability Solver,” *Proc. of ICCAD 2001*, Nov. 2001.