A ROBDD-Based Generalized Nodal Control Scheme for Standby Leakage Power Reduction

Hsinwei Chou

Dept. of Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, WI 53706
hsinweichou@wisc.edu

Charlie Chung-Ping Chen

Department of Electrical Engineering
National Taiwan University
Taipei, 106, Taiwan
cchen@cc.ee.ntu.edu.tw

Abstract— We present in this paper the method of Generalized Nodal Control for standby leakage power reduction. First, a sparse set of input and internal nodes to control are identified using a novel algorithm based on Reduced Ordered Binary Decision Diagrams (ROBDDs) and Fiduccia-Mattheyses Partitioning. Then, customized control gates with "forced" transistor stacks are inserted at these select locations to achieve effective subthreshold leakage control with minimal overhead addition. Compared to the method of input assignment, our technique is more generalized and more effective, especially on large circuits with high logic depths. We implemented our algorithm in C and tested it on several ISCAS85 and MCNC91 benchmark circuits. Experimental results show an average of 29% standby leakage power reduction and a worst runtime of 3 minutes only.

I. Introduction

As CMOS scales into nanometer technology, the importance of standby leakage power reduction, especially for portable devices, cannot be underestimated. In past works, several leakage reduction methods have been proposed, such as dual Vt[1], MTCMOS[2], and input assignment[3][4][5][6]. Each technique has its share of strengths and drawbacks, and in this work we are concerned only with the last approach, input assignment. The key idea behind input assignment is to explicitly control the input nodes' values during STATIC or standby mode operation so that the target circuit is placed into a 'leakage-optimal state' during the entire time that it idles. We will elaborate more on this in Section II. Note that this technique only works for combinational circuits, since the internal nodes' values get overwritten whenever a new input vector is applied.

Compared to other leakage reduction methods like dual Vt and MTCMOS, the main appeal behind input assignment lies in its low overhead, as the only penalty incurred in using this technique comes from the additional hardware needed for input pin control (which can be as simple as AND/OR gates or latches/muxes). In addition, the number of inputs to a circuit is typically limited, which further alleviates the overhead penalty. The disadvantage of input assignment is that as a circuit's size gets larger and larger, the effectiveness of input assignment tends to degrade. This is because having more gates in a circuit increases the number of variables (the leakage states of the individual gates) to optimize for, which in turn makes the finding of the optimal input vector more difficult. Clearly, this is a strong drawback to input assignment, as designs are constantly getting bigger with technology scaling.

In light of the inefficiency of input assignment in large circuits, we propose in this work a new method for leakage power reduction called Generalized Nodal Control, where the idea is to allow any node in the circuit to be controllable, regardless of whether it's an input node or an internal node. In this manner, the points of control within a circuit can be distributed more evenly for finer leakage management. However, the key difficulty involved is how to decide which and how many nodes to control such that a satisfactory amount of leakage reduction can be attained without adding too much overhead penalty. As we will show in Section III, it turns out that a good answer to this question can be found by taking a divide-and-conquer approach and using Reduced Ordered Binary Decision Diagrams (ROBDDs)[10] to solve the optimal input vector identification problem. However, before we elaborate on the details of our technique, we first provide some background information in Section II on input assignment. Then, after developing the main Generalized Nodal Control scheme in Section III, we present some experimental results in Section IV, followed by concluding remarks in Section V.

II. Background

In general, the leakage current within a transistor is comprised of several components like subthreshold leakage, gate tunneling leakage, reverse-biased pn junction leakage, etc. For a more detailed analysis, we refer the readers to[7]. For this paper, only a high-level view is needed, which is the following observation: due to the way the transistors are stacked on top of each other in CMOS standard cells, an important phenomenon known as the stacking effect[8] causes a CMOS gate's subthreshold leakage power consumption to be sensitive to its input combination value. For example, the leakage characteristics for several basic CMOS cells are tabulated in Table I. As it can be seen, i_{leak} can vary non-

TABLE I
Leakage current (e-9) corresponding to different input combinations (70nm technology, TOX=12nm, 110° C)

INPUTS	NAND2	NAND3	NAND4	NOR2	NOR3	NOR4
0000	151	85	58	1293	1929	2552
0001	644	151	85	270	256	248
0010	513	152	85	386	267	253
0011	784	639	151	57	54	53
0100		147	85		381	265
0101		510	151		57	54
0110		496	151		56	54
0111		1174	634		29	29
1000			83			375
1001			144			57
1010			145			56
1011			507			29
1100			143			56
1101			493			29
1110			486			29
1111			1562			20

trivially between different input patterns. In general, the leakage power consumption of any simple CMOS gate differs with respect to different input combinations, and thus a CMOS circuit as a whole is input vector-dependent for its leakage power consumption. Hence, as was discussed earlier, input assignment involves pre-determining a set of input values which is optimal (in the leakage power sense) for a given target circuit and inserting special hardware to force these values to be applied automatically during STATIC mode. Unfortunately, the problem of finding the 'best' input vector is known to be NP-complete. Nevertheless, several heuristics for solving this problem have been proposed in recent works[5][6][12][13].

III. Generalized Nodal Control

We now present our algorithm for Generalized Nodal Control. To be able to achieve satisfactory leakage control without incurring too much overhead penalty, we must not only identify a good set of nodes to control, but also minimize the total number of elements in that set. Since there exists 2^n (n = # of nodes) total possible subsets of nodes in a circuit, we can clearly not enumerate all possible ways to control the nodes to find the best choice. Instead, we need a reasonably good approximation heuristic to do this, and to this endeavor we propose a strategy based on divide-and-conquer with 'multiple' input assignments. Our method is an intuitive extension of input assignment and works as follows: given a target circuit whose size may be too large for effective input vector control, we first partition the circuit into as many sub-circuits as it takes to achieve decent leakage control on each sub-circuit, then optimally and independently identify an input vector for each sub-circuit as if they were separate entities. Finally, we add in some nodal-control hardware at selective locations where cuts occurred so that all of the pre-determined input vectors for each sub-circuit can be properly enforced during STATIC mode. In doing this, we have now extended the points of control from input-only to inputs and internal nodes as well (since the inputs to a sub-circuit can come from either an actual primary input or an internal node belonging to the cut set formed from partitioning).

By allowing input assignment to be applied independently on each smaller sub-circuit, our Generalized Nodal Control scheme is guaranteed to achieve leakage control at least as good as that of the original input assignment method, if not better. However, one can easily imagine that when the number of partitions get too large, inserting nodal-control hardware at every node in the cut set can be costly on overhead penalty. Fortunately, there is a way to get around this problem, and it involves noticing a key trend present in the data shown in Table I. Observe that for all of the gates shown, i_{leak} can vary substantially between some input combinations and yet trivially between others. For example, in the NAND4 case, the subthreshold leakage values for input combinations '0000' and '0100' are approximately equal, relatively speaking. Because of this, we can say without loss of generality that if a NAND4 gate sees an input combination of '0X00', where X is a 'don't care', then its leakage current will be approximately equal to some value, say the average between '0000' and '0100'. By exploiting the natural existence of 'don't care' variables between similar leakage input combinations, we can significantly reduce the number of input nodes needed to be controlled for optimal input assignment in each of the sub-circuits. We call this idea 'partial' input vector control to distinguish it from the standard input assignment approach, where all input values are explicitly controlled to achieve the absolute-best possible leakage state (even if it means sacrificing some overhead in adding an additional node to control in exchange for minor savings in leakage power consumption). To the best of our knowledge, all heuristics proposed to-date for optimal input vector identification have been concerned only with full input vector assignment. As such, they were not suitable for this work and we needed to develop a new method to determine the optimal input vector such that the concept of 'partial' input control can be realized. For this, we turn to ROBDDs, an idea that will be elaborated in Section III-B.

The following sub-sections describe the main Generalized Nodal Control scheme, which is divided into three key phases: partitioning, partial input vector identification, and merging.

A. First Phase: Partitioning

Given a target circuit, we first partition it into as many sub-circuits as we desire using a classical partitioning algorithm in Multilevel Fiducia-Metheyes (MLFM)[9]. In MLFM, balanced partitions are sought using as few cuts as possible. This goal matches favorably with our technique, since by maintaining a size balance amongst the different partitions, each resulting sub-circuit will be equally likely to be tractable. Furthermore, the fewer the cuts, the lesser the number of internal nodes that need to be controlled in the end, and therefore the smaller the overhead penalty. The number of ways to partition the circuit is chosen based on the size of the initial circuit and on the desired level of leakage control by the user. In general, more partitions result in finer leakage management, but also generate a greater overhead at the end. It is up to the user to make the tradeoff between leakage power reduction and overhead penalty.

B. Second Phase: Partial Input Vector Identification

Once partitioning is complete, the next step is to independently identify a low-leakage partial input vector for each partition. To do so, every basic CMOS gate contained in the cell library must have its leakage-input relation precharacterized via SPICE like that shown in Table I. This is necessary to introduce leakage 'tiers' for each gate type. A leakage tier is defined as a small and distinct range of leakage values. For example, using Table I, a NAND3 can be characterized to have 3 tiers: one with {'000', '001', '010', '100'}, another with {'011', '101', '110'}, and the last being {'111'}. An important point to note is that for certain types of gates like inverters and XORs/XNORs, only one leakage tier is exhibited. For these cells, it just means that their leakage power consumption is not affected by their input combination value, and hence a method for finding a minimal-leakage input vector need not directly take these gates into account.

The way in which a tier is defined or grouped affects how partial the end-resulting input vector will be. The looser the grouping(fewer total leakage tiers for that particular gate type), the more partial the result will be, since more 'don't care' variables can surface. However, a looser grouping translates to a solution which is inferior in accuracy of leakage control to a solution which is found using a tighter grouping. It is up to the user again to pick the tradeoff point in accuracy vs. partiality. If the number of ways a circuit was partitioned was large, then it makes sense to group looser to incur less overhead. However, if only a small number of partitions were used, then one should probably group tighter to exercise better leakage control. For this work, we found that 2 to 3 leakage tiers sufficed for all of the standard cells used.

After characterizing the leakage tiers for every cell, the next step is to introduce a 'LeakageClause' for every gate that has more than one total leakage tier. These gates represent those in the circuit whose leakage value varies nontrivially between different input combinations, and thus can actually benefit from input control. The LeakageClause models the leakage-input dependence of a gate and is formulated as follows: Suppose that gate$_i$ was pre-characterized to have N leakage tiers, T_1^i, T_2^i,... T_N^i. Let the following notation comb$_i^j$ be used to denote an input combination that corresponds to leakage tier T$_i$, with j being just an index to distinguish between different combinations of the same leakage tier (i.e., comb$_1^1$, comb$_1^2$, and comb$_1^3$ are all input combinations that result in T$_1$). Then, the general form of a LeakageClause$_i$ can be written as follows (note that "+" is the Boolean OR operation):

$$\begin{aligned} LeakageClause_i \quad = \\ \left[comb_1^1 + comb_1^2 + ... + comb_1^j\right](T_1^i)(\overline{T_2^i})...(\overline{T_N^i}) \quad + \\ \left[comb_2^1 + comb_2^2 + ... + comb_2^j\right](\overline{T_1^i})(T_2^i)...(\overline{T_N^i}) \quad + \\ ... \\ \left[comb_N^1 + comb_N^2 + ... + comb_N^j\right](\overline{T_1^i})(\overline{T_2^i})...(T_N^i) \end{aligned} \qquad (1)$$

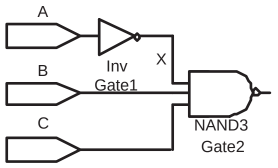

Fig. 1. Example Circuit

An important thing to note is that in a LeakageClause, the input combination expressions comb$_i^j$ are formed from the previous gates' logic functions, which are recursively expressed down to only the input variables. This is done to avoid the introduction of internal node variables while formulating the LeakageClause. For example, LeakageClause$_2$ for the circuit shown in Figure 1 is written as follows (assuming the same NAND3 tier characterization as earlier):

$$\begin{aligned} LeakageClause_2 \quad = \\ \left[(A)(\overline{B})(\overline{C}) + (A)(\overline{B})(C) + (A)(B)(\overline{C}) + (\overline{A})(\overline{B})(\overline{C})\right] \\ (T_1^2)(\overline{T_2^2})(\overline{T_3^2}) \quad + \\ \left[(A)(B)(C) + (\overline{A})(\overline{B})(C) + (\overline{A})(B)(\overline{C})\right] \\ (\overline{T_1^2})(T_2^2)(\overline{T_3^2}) \quad + \\ \left[(\overline{A})(B)(C)\right](\overline{T_1^2})(\overline{T_2^2})(T_3^2) \end{aligned}$$

Observe that in the above equation, the first input to the NAND3, node X, is implicitly replaced with $(\overline{A})$ in the input combination expressions.

After deriving the LeakageClause for every gate with more than one leakage tier, the next step is to AND them all together to form a conjunction called the CircuitLeakage.

$$CircuitLeakage = \Pi_{i=1}^{n}(LeakageClause_i) \qquad (2)$$

It is important to realize that the CircuitLeakage equation is always satisfiable because for any particular input vector assignment, the values for the leakage tier variables, T_j^is, can be arbitrarily set to match the actual resulting tier states for the corresponding gates in the circuit under that input vector.

Once CircuitLeakage is derived for the target circuit, a 'weighted' ROBDD[10], which we will call the FinalBDD, can then be constructed to model the CircuitLeakage equation. It is important to note that arc weights are not traditionally present in ROBDDs, so this is a minor variation of the standard data structure. The weights are assigned to the arcs of FinalBDD as follows: For those nodes associated with the primary inputs, both their THEN and ELSE arcs are arbitrarily assigned a small and non-zero value. The non-zero property on these weights is very important for partial input vector identification and will be explained in more details soon. For the leakage tier nodes T_j^is, their ELSE arcs all have a weight of 0 while their THEN arcs have a weight that is proportional to the leakage value represented by that tier. If a leakage tier is associated with more than one input combination, then its THEN arc's

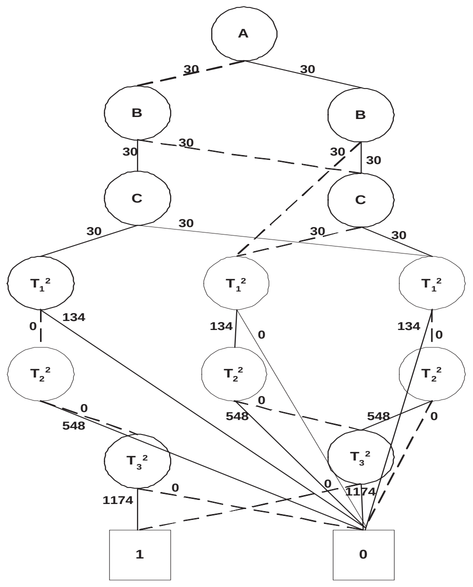

Fig. 2. FinalBDD for Figure 1. Solid lines=THEN, dashed=ELSE

weight is computed as the average of those input combinations' resulting leakage values. For example, suppose we are dealing with a NAND3 gate. Then, using the same NAND3 leakage tier characterization as earlier, we would assign its T_1 node's THEN arc a weight of 134, which is the average between 85('000'), 151('001'), 152('010'), and 147('100'). Similarly, its T_3's THEN arc would have a weight of 1174('111'). As an illustration, Figure 2 shows the FinalBDD for the circuit of Figure 1.

After constructing FinalBDD, we now have a Directed Acyclic Graph (DAG). From the FinalBDD DAG, a low-leakage partial input vector can be determined by tracing the shortest path from root to the logic-TRUE node (the '1' node on the bottom), where shortest is defined as min(Σ all arc weights in path) for all possible paths to TRUE. That is, finding an optimal partial input vector is equivalent to solving the shortest path problem in FinalBDD. Through this path, by examining which of the arcs (THEN or ELSE) are taken from the input nodes, the values to control the different input signals can be determined. Furthermore, those inputs which do not need to be controlled can be identified and left alone in the final design. For example, in Figure 2, the shortest path is $(\text{A})(\overline{B})(\text{T}_1^2)(\overline{T_2}^2)(\overline{T_3}^2)$, so the optimal-leakage partial input vector is simply AB = {10}. Notice that input C is not included in the solution because regardless of what C's value is, as long as AB = {10}, the leakage state of this circuit will be considered 'low enough' by the way we defined the leakage tiers.

At this point, one may wonder why we chose to use a ROBDD as our graph data structure for CircuitLeakage when it appears that just a simple binary tree would suffice for the method described. There are actually two main reasons for this. First, the number of nodes in the graph is exponential with respect to the number of gates in the circuit, due to the fact that each gate introduces multiple leakage tier nodes. Hence, if a regular binary tree were used, the size of the graph can easily explode out of control. Fortunately, the ROBDD data structure is an actively researched topic and past works have led to an attractive mechanism called dynamic variable reordering[11] which can significantly compress the size of a ROBDD (hence the label 'Reduced'). This is the first main reason behind the choice of ROBDD. Second, although it's not shown in the example graph of Figure 2, there are times when even the shortest path solution can be sub-optimal in the sense that even though the controlled leakage state may be low, the number of inputs required to be controlled may not be absolutely minimized. This is because depending on the way the node variables are ordered in the graph from top to bottom, a path from root to TRUE can sometimes unnecessarily traverse 1 or more input node(s). For instance, in the ROBDD of Figure 2, node A is the first node from root, so it must be traversed no matter what. However, it is possible that if the tree had a different node ordering where A is not the first node from the root, then we could potentially bypass A in tracing a shortest path to TRUE. Therefore, the order of the nodes can play a large role in determining how 'partial' our final input vector gets. Hence, a fixed-order, static binary tree would not be judicious to use, and the dynamic variable reordering mechanism of ROBDDs again comes into play. From methods like Rudell's sifting algorithm to Fujita's window algorithm (see [14]), many different ways to dynamically reordering a ROBDD can be tried until we pick out the best and most 'partial' input vector solution.

Recall that we previously said that the arc weights of the input variable nodes should be small but non-zero. The reason for this is that in assigning small weights to the input nodes (with respect to the weights on the tier state nodes), we can allow a shortest path finding procedure like Dijkstra's Algorithm to prioritize for leakage reduction over the minimization of the number of input nodes to control. This is because the weights on the tier state nodes will dominate those on the input variables' arcs during the process of determining the shortest path. However, because the weights of the input nodes are non-zero, it will still allow the number of input nodes traversed in the final solution to be minimized in the end, as long as the ending leakage state remains unchanged.

The entire partial input vector identification process using ROBDDs is now summarized in Algorithm 1.

C. Third Phase: Merging

After we have identified a low-leakage partial input vector for each partition, the last step in our algorithm is to 'merge' these results together and determine which nodes need to be controlled via an insertion of a 'switch'. A switch is just a gate which serves to output a fixed logic value during STATIC mode. In normal ACTIVE mode, a switch will behave like an ordinary buffer. For example, an AND gate with the STATIC_COMPLEMENT signal as one of its inputs is a switch for forcing a logic value of 0 during STATIC mode. Similarly, an OR gate with the STATIC signal as one of its inputs is a mechanism for logic 1-forcing.

```
Enable automatic dynamic variable reordering;
FinalBDD = the constant-1 ROBDD;
for all gates in the circuit do
    /* Assume that the output of this gate is node i
    and its inputs are nodes a, b, ... , m */
    if current gate has more than 1 leakage tier then
        LeakageBDD$_i$ = BuildBDD(LeakageClause$_i$);
        Assign arc weights to LeakageBDD$_i$;
        FinalBDD = $\prod$ (FinalBDD)(LeakageBDD);
    end
end
BestShortestPath = NULL;
$\alpha$ = user-specified threshold limit;
while $\alpha \geq 0$ do
    TempPath = ShortestPathTo1(FinalBDD);
    if Length(TempPath) < BestShortestPath then
        BestShortestPath = TempPath;
    end
    $\alpha$ = $\alpha$ - 1;
    Dynamically reorder the nodes of FinalBDD using
    a different variable reordering heuristic;
end
Return BestShortestPath;
```

Algorithm 1: Partial Input Vector Identification

Switches should not just be inserted at every internal node belonging to the cut set. To minimize the overhead penalty, we only insert a switch at an internal node if its 'natural' STATIC mode input and output values do not match. That is, consider a cut occurs on node x. Then, x will be an output in one partition, let's say partition A, and an input in another, let's say partition B. To determine whether a switch should be inserted at node x, we first find out what the 'natural' logic value for x will be during STATIC mode under the pre-identified partial input vector solution for partition A. Then, we compare this value with that which is required by partition B's partial input vector solution. If the values match or if there is a 'don't care' in B's partial input vector at the corresponding input node location, then we don't need to insert a switch at x because the values can 'naturally' converge. If they don't, then a switch is needed at x to force out its opposite value during STATIC mode. This is because the value of x needed to compose B's partial input vector solution will be different from that which is normally outputted by A under A's partial input vector solution.

Care must be taken when actually implementing a switch, for we have empirically found that if the switches were implemented in normal CMOS configuration or as basic latches/MUXes, then the resulting leakage consumption of the switches themselves can easily make up for much of the leakage reduction benefit gained from generalizing nodal control. The proper way to implement a switch is to modify a standard CMOS AND/OR gate so that it has transistors 'forcefully' stacked. For more information on this method, we refer the readers to [8]. Figure 3 shows our logic 0-forcing switch. The logic 1-forcing switch can be derived in a similar manner, but using an OR gate instead of an AND gate.

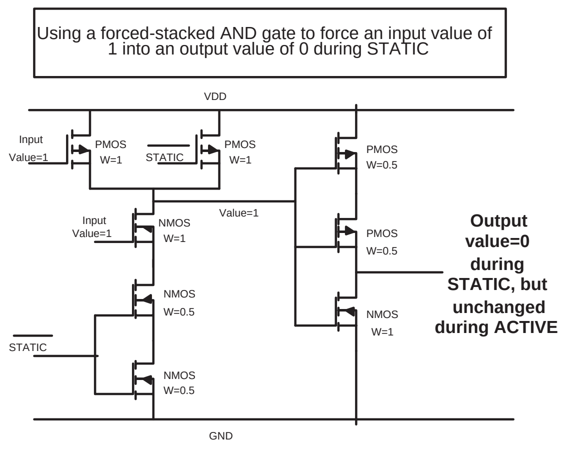
Using a forced-stacked AND gate to force an input value of 1 into an output value of 0 during STATIC
VDD
Input Value=1
PMOS W=1
STATIC
PMOS W=1
PMOS W=0.5
Input Value=1
NMOS W=1
Value=1
PMOS W=0.5
Output value=0 during STATIC, but unchanged during ACTIVE
NMOS W=0.5
NMOS W=1
STATIC
NMOS W=0.5
GND

Fig. 3. Low-Leakage Logic 0-Forcing Switch

The formulation of our Generalized Nodal Control scheme is now complete and can be summarized as follows: given a target circuit, first partition it into a set of suitably small sub-circuits, then individually identify an optimal 'partial' input vector for each partition using ROBDDs. Finally, perform input-output compatibility analysis at all internal nodes to carry out judicious switch insertion.

D. Time Complexity of Generalized Nodal Control

The time complexity of our algorithm is O(N log N) for the shortest-path finding procedure via Dijkstra's Algorithm. However, since N, the number of nodes in FinalBDD, is exponential with respect to the number of leakage tiers introduced for each gate, we actually end up with an exponential time complexity (i.e., $O(3^n \log 3^n)$ if 3 leakage tiers are used for each gate type, which is the case in this work). Little n in this case denotes the number of gates in the partition. This exponential time complexity more or less agrees with the NP-completeness of the input assignment problem. Nevertheless, since we can always reduce n in the above equation by partitioning more, the actual runtime of our method is actually not as bad as it appears. Also, instead of Dijkstra's algorithm, a faster shortest-path-finding algorithm could have been used to reduce the runtime as well.

IV. Experimental Results

We conducted all of our experiments on a machine with 320MB of Ram and running on an Intel P2 600Mhz processor. 70nm technology was used in our SPICE simulations. All algorithms were implemented in C, and two off-the-shelf packages, one ROBDD[14] and one MLFM partitioner[15], were used in this work. A meaningful set of circuits taken from the ISCAS85 and MCNC91 benchmarks were used for experimental tests.

TABLE II
Leakage Current Reduction Comparisons

	i_{leak} (uA) Original	Num. of Partitions	i_{leak} (uA) GNC	% i_{leak} Reduction	Runtime (secs)
i6	235.78	15	185.27	21.4%	23
i7	310.97	20	178.86	42.5%	41
i8	596.69	40	386.90	35.2%	93
i9	245.02	25	131.44	46.4%	23
i10	1110.00	80	800.57	27.9%	190
c2670	309.82	20	237.98	23.2%	20
c3540	530.82	40	398.12	25.0%	61
c5315	715.96	50	572.99	20.0%	109
c6288	1430.00	80	1060.00	25.9%	90
c7552	1080.00	50	828.56	23.3%	79

TABLE III
Area, Delay, and Dynamic Power Overhead Additions

	Area Original (SIS)	Area GNC (SIS)	Delay Original (SIS)	Delay GNC (SIS)	Dynamic Power Original	Dynamic Power GNC
i6	592528	599488	9.11	11.2	7.22e-4	7.34e-4
i7	691360	734512	9.95	12.18	9.28e-4	1.00e-3
i8	1309872	1485264	15.78	19.22	2.23e-3	2.54e-3
i9	629184	747504	16.10	18.29	8.62e-4	9.87e-4
i10	2730176	3126896	49.01	56.78	3.74e-3	4.34e-3
c2670	876496	960016	23.67	30.48	1.41e-3	1.55e-3
c3540	1328432	1556720	40.71	49.28	2.44e-3	2.84e-3
c5315	2072688	2294016	33.53	38.72	3.60e-3	3.99e-3
c6288	4069280	4319840	112.91	129.63	1.43e-2	1.67e-2
c7552	2895824	3163088	36.98	42.47	6.78e-3	7.75e-3

Table II shows the leakage reduction effectiveness of GNC, short for Generalized Nodal Control. α (from the partial input vector identification algorithm) was arbitrarily set to 10. Only one arbitrary choice of the partition count is shown due to space limitation. The results indicate that GNC can achieve a moderate amount of leakage reduction with a very low runtime, regardless of the size of the target circuit. Of course, adding internal node switches naturally introduces some overhead[1], so we show just how much this amounts to in Table III (HSPICE and SIS[16] were used). As it can be seen, this overhead is kept relatively low due to the 'partialness' of input vectors as well as the selectiveness of switch insertion.

Some heuristics could have probably been used to reduce the penalty shown in Table III. For example, one could try to insert switches only on the non-critical paths. However, we refrained from doing so because one, the overhead does not appear to be overly excessive in the first place, and two, if such heuristic were used, then the optimality of the result will depend significantly on the order in which the partitions were processed, since the slack and the internal node values must be updated in-between each partition processing. At this time, it is not clear to us how to best determine the optimal order without degrading the runtime significantly, so we leave this matter for a future work.

Finally, Table IV shows that our technique is scalable for circuits of arbitrarily large size. As it can be seen, larger circuits merely require more partitions to be formed in order to keep the runtime and leakage reduction effectiveness satisfactory.

[1]It should be noted that this penalty is present in the method of input assignment as well, since the switches themselves inevitably contribute some overhead to area, dynamic power consumption, and delay(if inserted on the critical path).

TABLE IV
C6288: varying partition count vs. reduction and overhead

	Num of Switches	i_{leak} GNC (uA)	Runtime (secs)	% Area Increase	% Delay Increase	% D. Power Increase
60	236	1140	104	5.4 %	13.9 %	15.14 %
80	262	1060	90	6.1 %	14.8 %	16.7 %
100	289	1013	84	6.7 %	14.6 %	18.22 %
120	314	946	80	7.3 %	16.4 %	20.01 %

V. Conclusion

In this paper, we have presented a novel, decision diagram-based leakage reduction technique called Generalized Nodal Control. Experimental results demonstrate the viability of our approach, with our technique achieving on average close to 30% leakage power reduction with a worst runtime of 3 minutes only.

VI. Future Work

The Generalized Nodal Control scheme described in this paper attempts to control any node in the circuit as long as it leads to a good amount of leakage reduction. As a result, the final critical path delay can worsen due to the insertion of switches along nodes in the critical path. If it is absolutely critical to keep the worst path delay from increasing, then one can apply the following alternative version of Generalized Nodal Control, which we are currently working on for a future work: the idea is that in phase 2 of the algorithm where we identify the optimal partial input vector, for every one of those input nodes which belong to the critical path, we assign a weight of infinity to both its THEN and ELSE arcs to try to force the final shortest path solution to not involve the control of these nodes. Also, when applying input assignment to each of the individual partition, we must do this in a depth-first manner so that the slack can be guaranteed to be correct for all of the nodes at any point in the algorithm. We start with the partition with the lowest depth, identify its optimal input vector and those nodes which need to be controlled, update the delay slack at all of these nodes due to switch insertion, then continue on with input vector identification for the next deepest partition. This alternative approach will result in no increase in the final critical path delay, although the degree of leakage control is most likely going to degrade due to the new restrictions placed on which nodes are controllable and which are not. We plan to investigate this technique more in an upcoming work.

References

[1] T. Karnik, Y. Ye, et al., "Total power optimization by simultaneous dual-Vt allocation and device sizing in high performance microprocessors," in *Design Automation Conference*, 2002, pp. 486–491.

[2] S. Shigematsu et al., "A 1-V high-speed MTCMOS circuit scheme for power-down applications," in *Proc. IEEE Symp. VLSI Circuits Dig. Tech. Papers,* 1995, pp. 125–126.

[3] Y. Ye, S. Borkar, and V. De, "A new technique for standby leakage reduction in high-performance circuits," in *Intl. Symp. VLSI Circuits Dig. Tech. Papers,* 1998, pp. 40–41.

[4] D. Lee and D. Blaauw, "Static leakage reduction through simultaneous threshold voltage and state assignment," in *Proceedings of the Design Automation Conference,* 2003, pp. 191–194.

[5] M. C. Johnson, D. Somasekhar, and K. Roy, "Models and algorithms for bounds on leakage in CMOS circuits," in *IEEE Trans. Comput.-Aided Design Integrated Circuits Sys.,* 18:714–725, June 1999.

[6] F. Aloul, S. Hassoun, K. Sakallah, and D. Blaauw, "Robust SAT-based search algorithm for leakage power reduction," *International Workshop on Power and Timing Modeling, Optimization and Simulation,* Sevilla, Spain, 2002.

[7] K Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," in *Proceedings of the IEEE* 91(2):305–327, February 2003.

[8] S. Narendra, S. Borkar, et al., "Scaling of stack effect and its application for leakage reduction," in *Intl. Symp. Low Power Electronics and Design.,* 2001, pp. 195–200.

[9] S. Sait and H. Youssef, *VLSI Physical Design Automation.* World Scientific, 1999.

[10] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," in *IEEE Trans. On Computers,* 35(8):677-691, 1986.

[11] E. Felt, G. York, R. Brayton, and A. Sangiovanni-Vincentelli "Dynamic variable reordering for BDD minimization," in *Design Automation Conference,* 1993, pp. 130–135.

[12] A. Ferre and J. Figueras, "Leakage power bounds in CMOS digital technologies," in *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems,* Vol. 21, pp. 731–738, June 2002.

[13] J. P. Halter and F.N. Najm, "A gate-level leakage power reduction method for ultra-low-power CMOS circuits," in *IEEE Custom Integrated Circuits Conf.,* 1997, pp. 475–478.

[14] CUDD: http://vlsi.colorado.edu/∼fabio/CUDD

[15] hMETIS: http://www-users.cs.umn.edu/∼karypis/metis/hmetis

[16] SIS: http://www-cad.eecs.berkeley.edu/Software/software.html