An Algebraic Decision Diagram (ADD) Based Technique to find Leakage Histograms of Combinational Designs

Kanupriya Gulati Nikhil Jayakumar Sunil P. Khatri

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

ABSTRACT

In this paper, we present an Algebraic Decision Diagram (ADD) based approach to determine and implicitly represent the leakage value for all input vectors of a combinational circuit. In its exact form, our technique can compute the leakage value of each input vector. To broaden the applicability of our technique, we present an approximate version of our algorithm as well. The approximation is done by limiting the total number of discriminant nodes in any ADD. Previous sleep vector computation techniques can find either the maximum or minimum sleep vector. Our technique computes the leakages for all vectors, storing them implicitly in an ADD structure. We experimentally demonstrate that these approximate techniques produce results which have reasonable errors. We also show that limiting the number of discriminants to a value between 12 and 16 is practical, allowing for good accuracy and lowered memory utilization.

Categories and Subject Descriptors: B.7.1 [Integrated Circuits]: VLSI

General Terms: Algorithms

Keywords: ADD, BDD, Leakage

1. INTRODUCTION

The total magnitude of leakage currents in a design depends strongly on the input vector applied. As a consequence, there has been much recent interest in techniques to find the input vector that results in the *lowest* leakage for the circuit. However, with leakage power increasing as a fraction of the total power of a design, it is no longer sufficient to simply find the input vector that minimizes circuit leakage. It is now more important to find the leakage for *all* input vectors (of course, the minimum leakage vector can also be found at the same time). When comparing candidate implementations of a design with the same minimum leakage values, we should prefer the design that has a leakage histogram with the largest number of input vectors contributing lower leakage values. This would not only minimize the leakage during the regular operation of the circuit, but also ease the task of finding a vector which results in minimum leakage. It was reported in [1] that the maximum leakage value of a design can be as high as 2.4× the minimum value (1.6× on average), again underscoring the importance of computing the leakage of *all* input vectors for implementations and choosing one with a favorable leakage histogram. This paper is the first to develop a technique that performs this function. We compute the leakage of the design for all input vectors by using an Algebraic Decision Diagram (ADD) [2, 3] based technique, allowing us to represent the leakage of a design implicitly and compactly.

The remainder of this paper is organized as follows: Section 2 discusses some previous work in this area. In Section 3 we describe our ADD based method to compute the leakage of an arbitrary mapped design. In Section 4 we present experimental results, while conclusions and future work are discussed in Section 5.

ISLPED'05, August 8–10, 2005, San Diego, California, USA

2. PREVIOUS WORK

The problem of finding the minimum leakage sleep vector for a combinational CMOS gate-level circuit has received some attention recently. In [4], the authors reported a genetic algorithm based approach to solve this problem. The authors of [5] introduce a concept called leakage observability, and based on this idea, describe a greedy approach to finding the optimal sleep vector. In [6], the authors present a greedy search based heuristic, guided by node controllabilities and functional dependencies. The work of [1] is based on an ILP formulation. It makes use of pseudo-Boolean functions which are incorporated into an optimal ILP model and a heuristic mixed integer linear programming method as well.

In [7], the authors use ADDSs to find the leakage of a channel-connected region (CCR) as a function of its inputs. The focus in [7] was on full-custom circuitry and the authors used their technique to find functional failures in CCRs due to excessive leakage (input vectors that caused leakage to go above a certain value). Exclusivity constraints were added to constrain the ADD of a CCR to legal input vectors.

In our approach, we use Algebraic Decision Diagrams (ADDs) [2, 3] to *implicitly* represent the leakage of a standard-cell based design for *all* input vectors in a single structure. We use pre-characterized data (leakage values) from the standard-cell library to construct this *leakage ADD*. The inherent sharing of nodes in such a structure allows us to represent the leakage of the design compactly. In order to improve the efficiency of the leakage ADD construction, we

also bin the values of the leaf nodes to reduce the number of leaf nodes of the ADD. This reduces the number of discriminants[1] (as well as the number of nodes) in the leakage ADD of the design.

3. OUR APPROACH

Our approach, based on an ADD [2, 3] computation, is able to compute the leakage for all possible input vectors in the design. This is useful in several contexts, such as

- It allows us to compute the average, minimum and maximum leakage for the design in an accurate manner.
- It allows us to construct the histogram of leakage values for a design. This can be of use when comparing two or more candidate implementations (with similar maximum leakage values) of a single circuit. The design with a leakage histogram that is skewed toward the lower leakage values would be preferred, since it would reduce power under operating situations. For example, during operation, the circuit may switch between repeatedly between a set of vectors. In this case, the implementation which has a leakage histogram skewed as mentioned above would be preferred. Figure 1 illustrates this idea. The leakage histograms of two designs (with similar maximum leakage values) is shown. The histogram to the right is preferred, since it has a large number of vectors with low leakage values.

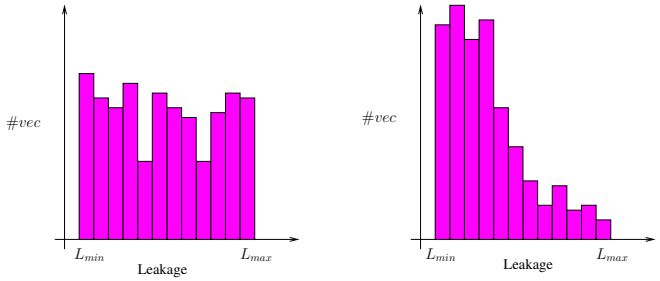


Figure 1: Leakage Histograms for two Implementations of a Design

3.1 Exact Computation of Leakages of all Vectors

Consider a combinational logic network η, consisting of logic gates H_j selected from some library P. We refer to the ROBDD of H_j as h_j, and the leakage ADD of H_j as $\mathcal{H}_j$. This ADD represents the leakage value of each minterm m of H_j obtained by following the path (from the root) indicated by the literals of m, until a terminal vertex is reached. The value of this vertex is the leakage of H_j under the input m.

We assume that for each gate H_j, we have an array called ($lkg\_array(H_j)$) describing its leakage values for all possible values of its immediate fanins. For example, the if H_j was a 2-input gate, then its leakage array would consist of 4 values, corresponding to all 4 possible input combinations for the gate. Let us assume that the 2 fanins are called F and G. We assume for ease of the exposition that we have these sorted in numerical order, so that the leakage value of the input combination 00 appears first, followed by that of the input values 01, and so on. Suppose that under some minterm m, the ROBDDs f and g evaluate to f_{val} and g_{val} respectively. The corresponding leakage value for the gate H_j is found by indexing the $(f_{val} : g_{val})^{th}$ value of $lkg\_array(H_j)$. For example, if $f_{val} = 1$ and $g_{val} = 0$, then we index the second value of $lkg\_array(H_j)$ to obtain the appropriate leakage value.

Our algorithm first finds the ROBDDs of all network nodes. Next, we find the (global) leakage ADDs of each of the nodes in the network using Algorithm 1. Suppose we want to compute the leakage ADD of H. Assume that it has 2 fanins F and G. The leakage ADD of H is found by the subroutine $node\_compute\_lkg\_ADD(f, g, lkg\_array(H)$. In this routine, if the ROBDDs f and g are constant (f_{val} and g_{val} respectively), then the leakage value for this condition is simply found by indexing the $(f_{val} : g_{val})^{th}$ value of $lkg\_array(H)$ and returning an ADD node of this value. If either of f or g are non-constant, then we find the top variable among these ROBDDs, and recursively compute $\mathcal{H}_v$ and $\mathcal{H}_{\overline{v}}$ and return $\mathcal{H} = ITE(v, \mathcal{H}_v, \mathcal{H}_{\overline{v}})$.

Algorithm 1 The node_compute_lkg_ADD Algorithm

```
node_compute_lkg_ADD(f, g, lkg_array(H))
// terminal case below
if f_val = is_constant(f) && g_val = is_constant(f) then
   H = create_ADD_node(f_val : g_val)
   return H
end if
v = topvar(f, g)
f_v = cofactor(f, v)
f_v̄ = cofactor(f, v̄)
g_v = cofactor(g, v)
g_v̄ = cofactor(g, v̄)
H_v = node_compute_lkg_ADD(f_v, g_v, lkg_array(H))
H_v̄ = node_compute_lkg_ADD(f_v̄, g_v̄, lkg_array(H))
H = ITE(v, H_v, H_v̄)
return H
```

Note that the Algorithm 1 is applicable for gates H_j with two inputs. Our technology library consisted of at most 4-input gates. As a result, we require two additional routines similar to Algorithm 1, for 3 and 4 input gates.

Note that leakage ADDs of the mapped gates of the network need not be computed in any particular order. After the leakage ADDs of each gate have been computed, we find the leakage ADD of the entire circuit (we refer to this as $\mathcal{H}_{total}$), by adding each gate's leakage ADD. The routine to add two ADDs is shown in Algorithm 2. If the circuit has n gates, then this operation requires $n - 1$ addition operations, since we perform the addition of ADDs in a pair-wise manner.

Algorithm 2 first tests if the ADDs $\mathcal{F}$ and $\mathcal{G}$ to be added are both constants. If this is the case (call the constants $\mathcal{F}_{val}$ and $\mathcal{G}_{val}$) it creates and returns an ADD node with value $\mathcal{F}_{val} + \mathcal{G}_{val}$. If at least one of $\mathcal{F}$ or $\mathcal{G}$ are non-constant, then we find the top variable v among them. We recursively compute $\mathcal{H}_v = add\_ADD(\mathcal{F}_v, \mathcal{G}_v)$ and $\mathcal{H}_{\overline{v}} = add\_ADD(\mathcal{F}_{\overline{v}}, \mathcal{G}_{\overline{v}})$, and return $\mathcal{H} = ITE(v, \mathcal{H}_v, \mathcal{H}_{\overline{v}})$.

Once we have computed $\mathcal{H}_{total}$, the sum of all the leakage ADDs of the gates in the design, we can find the minimum valued leaf L_{min} (which is the minimum discriminant of $\mathcal{H}_{total}$) of the final ADD. This discriminant corresponds to the lowest leakage state of the design. We find a pri-

[1]The number of discriminants of an ADD is the number of unique leaves of the ADD

Algorithm 2 The add_ADD Algorithm

$add\_ADD(\mathcal{F}, \mathcal{G})$
// terminal case below
if $f_{val} = is\_constant(\mathcal{F})$ && $g_{val} = is\_constant(\mathcal{G})$ **then**
  $\mathcal{H} = create\_ADD\_node(\mathcal{F}_{val} + \mathcal{G}_{val})$
  return $\mathcal{H}$
end if
$v = topvar(\mathcal{F}, \mathcal{G})$
$\mathcal{F}_v = cofactor(\mathcal{F}, v)$
$\mathcal{F}_{\overline{v}} = cofactor(\mathcal{F}, \overline{v})$
$\mathcal{G}_v = cofactor(\mathcal{G}, v)$
$\mathcal{G}_{\overline{v}} = cofactor(\mathcal{G}, \overline{v})$
$\mathcal{H}_v = add\_ADD(\mathcal{F}_v, \mathcal{G}_v)$
$\mathcal{H}_{\overline{v}} = add\_ADD(\mathcal{F}_{\overline{v}}, \mathcal{G}_{\overline{v}})$
$\mathcal{H} = ITE(v, \mathcal{H}_v, \mathcal{H}_{\overline{v}})$
return $\mathcal{H}$

mary input vector that results in this leakage value by using Algorithm 3. A similar exercise can be conducted for any discriminant, allowing us to build a leakage histogram.

Algorithm 3 Finding an Input Vector with Minimum Leakage L_{min}

$find\_a\_minterm\_with\_min\_leakage(\mathcal{H}_{total})$
$\mathcal{H}_{thresholded} = ADD\_threshold(\mathcal{H}_{total}, L_{min} + \delta)$
$h_{thresholded} = ADD\_to\_BDD(\mathcal{H}_{thresholded})$
return $BDD\_find\_minterm(h_{thresholded})$

Thresholding an ADD consists of the task of converting it into an ADD with fewer discriminants. $ADD\_threshold(\mathcal{H}, val)$ makes all discriminants with values greater than or equal to val point to the 0 discriminant. All discriminants with values less than val are retained in the result.

Algorithm 3 first thresholds $\mathcal{H}_{total}$ with the value $L_{min}+\delta$. The value δ is such that there is no leakage value for the design in the closed interval $[L_{min}, L_{min} + \delta]$. In other words, there is no discriminant in the leakage ADD $\mathcal{H}_{total}$ in the above closed interval. Therefore the resulting leakage ADD after thresholding ($\mathcal{H}_{thresholded}$) consists of exactly two discriminants (L_{min} and 0). Next, $\mathcal{H}_{thresholded}$ is converted into a BDD, by replacing the L_{min} discriminant by the 1 discriminant. We now find a path to the 1 terminal node in this BDD, by using the well known linear-time BDD algorithm to find a single minterm.

In a similar manner, we can find the BDD for any specific leakage value (i.e. any specific discriminant of the leakage ADD). For a general leakage value L other than the maximum or minimum, we need to do the thresholding with threshold values $L + \delta$ as well as $L - \delta$, where δ is such that there is no other discriminant of the leakage ADD in the interval $[L+\delta, L-\delta]$. From the resulting BDD of the result, we can use standard linear-time BDD algorithms to find the number of minterms for the discriminant of value L. From this, we compute the leakage histogram for the circuit.

We utilized the CUDD [8] package for all the ADD operations in this paper. This package has routines to perform the operations described in the algorithms described in this paper.

3.2 Approximate Computation of Leakages of all Vectors

In an exact ADD representation of circuit leakage, the number of discriminants can be quite large. As a consequence, it is important to compute the circuit leakage ADDs in an approximate manner. This enables us to reduce the memory consumption and allows us to handle larger designs.

3.2.1 Binning of Leakage ADD values

Since our library consists of gates with up to 4 inputs, the maximum number of discriminants for the leakage ADDs of any gate is limited to 16. However, when we perform the add_ADD operation on two ADDs with D_1 and D_2 discriminants, the resulting ADD after addition may have as many as $D_1 \cdot D_2$ discriminants. To control the size of the resulting ADD after addition, we perform a discretization of the discriminants of the result. The discretization is driven by a user-specified constraint k.

Consider the addition of two ADDs $\mathcal{F}$ and $\mathcal{G}$, using the add_ADD routine. Let the minimum and maximum discriminant values of $\mathcal{F}$ ($\mathcal{G}$) be $L^{\mathcal{F}}_{min}$ and $L^{\mathcal{F}}_{max}$ ($L^{\mathcal{G}}_{min}$ and $L^{\mathcal{G}}_{max}$) respectively. As a consequence, the minimum and maximum discriminant values of the result will be $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min})$ and $(L^{\mathcal{F}}_{max}+L^{\mathcal{G}}_{max})$ respectively. Let us refer to the interval between these two values as R. Now, we discretize the interval into k values $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min})$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{R}{k-1})$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{2R}{k-1})$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{3R}{k-1})$, $\cdots$, $(L^{\mathcal{F}}_{min} + L^{\mathcal{G}}_{min} + \frac{(k-2)R}{k-1})$, $(L^{\mathcal{F}}_{max} + L^{\mathcal{G}}_{max})$.

Now, during the terminal case computation of Algorithm 2, we compute $v = \mathcal{F}_{val} + \mathcal{G}_{val}$ and adjust its value to the nearest of the k discretized discriminant values described in the previous paragraph. Let us say that the adjusted value is v_{adj}. Then, the value returned by Algorithm 2 in the terminal case is v_{adj}.

This limits the total number of discriminants in the result of add_ADD to k, instead of $D_1 \cdot D_2$, resulting in significantly reduced memory utilization in general. Also, the maximum error introduced by a single step of this addition is $\frac{1}{2(k-1)}$, allowing the user to trade off the memory utilization and maximum error.

4. EXPERIMENTAL RESULTS

We applied our technique on a series of MCNC91 benchmark designs, using a 0.1μm technology library with 13 gates, with between 1 and 4 inputs. After running technology independent logic optimizations (*script_rugged* in SIS [9]), we mapped these designs for area and delay (again in SIS).

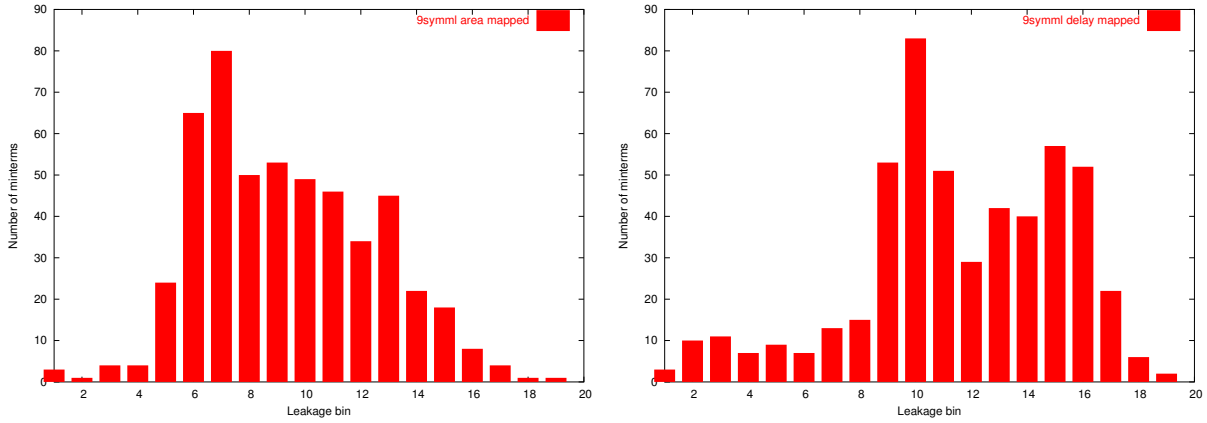
Our leakage computation technique was written in SIS, and implemented using the CUDD [8] package. When we applied our approximate technique with discretized discriminants, we were able to compute leakage ADDs for larger designs.

Table 1 describes the maximum and minimum leakages (in pA) of four designs, as a function of the value of k (the number of discretized discriminants we use during ADD construction). Each design was mapped for minimum area as well as minimum delay. The row labeled "exact" represents the leakages with no discretization of leakage values (effectively $k = \infty$). Note that a good choice of the values of k is between 12 and 16 for most cases.

We also computed leakage ADDs and computed the associated leakage histograms for some designs. Figure 2 shows

	9symml				cc				decod				alu2			
	Del		Area		Del		Area		Del		Area		Del		Area	
	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max
exact	622.9	734.6	474.1	611.8	193.2	272.5	127.2	227	187.8	238.6	30.6	79.9	1241.9	1382.9	872.8	1060.7
20 bins	540.8	772.3	429.1	633.2	209.6	267.8	131.5	221.1	200.8	239.1	31	83	905.5	1771.4	645.1	1348.5
16 bins	396.7	955.8	402.9	600.8	197.2	261.5	122	209.8	208	241.9	27.6	90.8	700.5	2005.2	576	1563.3
12 bins	285	1064.5	284	821.6	197.5	270.4	117.3	253.5	212.6	235.5	23.6	74.9	536.7	2193.2	484.8	1753.4
8 bins	212.4	1206.6	199.3	964.4	91	360.1	76.4	278	89.3	314.5	33	92.3	511.9	2251.2	382	1856.5
4 bins	212.4	1206.6	199.3	964.4	91	360.1	76.4	278	89.3	314.5	33	92.3	511.9	2251.2	382	1856.5

Table 1: Accuracy versus Bin Size



(a) 9symml-a

(b) 9symml-d

Figure 2: Leakage Histograms for Delay and Area Mapped Circuits

the histogram for one of the circuits. For this experiment, we used k=20. We found that area-mapped designs were typically "better", with a larger number of minterms having smaller leakage values.

5. CONCLUSION

In recent times, heuristic as well as exact approaches have been developed to compute the input vector which minimizes the leakage of a circuit. In this paper, we describe an approach to compute an ADD based implicit approach to find the leakage of all vectors in a circuit. The knowledge of the leakage of a circuit over all vectors can be used in several ways, one of which is to select between competing implementations of a circuit. Our approach computes the leakage ADDs of each circuit node, and then adds these ADDs to compute the leakage ADD of the circuit in terms of its primary inputs. We also implement an approximate version of this algorithm, which discretizes the number of discriminants of an ADD to a user-specified limit k. We experimentally demonstrate that these approximate techniques produce results which have reasonable errors. We also show that limiting the number of discriminant nodes to a value between 12 and 16 is practical, allowing for good accuracy and lowered memory utilization.

6. REFERENCES

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

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

[3] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large boolean functions with applications to technology mapping," in *Proceedings of the 30th international conference on Design automation*, pp. 54–60, ACM Press, 1993.

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

[5] M. Johnson, D. Somasekhar, and K. Roy, "Models and algorithms for bounds on leakage in CMOS circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, pp. 714–725, June 1999.

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

[7] H. Y. Song, S. Bohidar, R. I. Bahar, and J. Grodstein, "Symbolic failure analysis of custom circuits due to excessive leakage current," in *Proc. of the Intl. Conf. on Computer Design*, pp. 70–75, 2003.

[8] "CUDD: CU decision diagram package." http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html.

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