LOGIC SYNTHESIS USING POWER-SENSITIVE DON'T CARE SETS

by

Christopher K. Lennard, Premal Buch, and A. Richard Newton

Memorandum No. UCB/ERL M96/8

6 March 1996

LOGIC SYNTHESIS USING POWER-SENSITIVE DON'T CARE SETS

by

Christopher K. Lennard, Premal Buch, and A. Richard Newton

Memorandum No. UCB/ERL M96/8

6 March 1996

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Logic Synthesis Using Power-Sensitive Don't Care Sets*

Christopher K. Lennard

Premal Buch

A. Richard Newton

Department of Electrical Engineering & Computer Sciences

University of California, Berkeley, CA 94720

March 6, 1996

*This research was sponsored in part by the Semiconductor Research Corp. (SRC) contract 96-DC-324.

Abstract

The Boolean space spanned by the primary input vectors of a combinational function can contain a large variance in minterm probabilities. We show that it is possible to partition the boolean space into classes such that classes containing only 1% of the Boolean space cover as much as 40% of the total probability. In this paper, we exploit this characteristic of the minterm probability distribution to reduce power dissipation. The proposed technique uses the Don't Care set for optimizing the switching activity without compromising the flexibility for area optimization. Experimental results indicate that our method can yield low power realizations of the circuit without any penalty in area.

1 Introduction

During logic synthesis, the functionality of nodes internal to a circuit can be manipulated within the Don't Care (DC) set to minimize area or speed. Although a reduction in area often corresponds to a reduction in power, this is not true in CMOS technologies if the switching activity is increased. Design of a synthesis algorithm for reducing power consumption must take into account the fact that for circuits with distributed input switching probabilities, even a minimal Boolean difference in functionality at a node could dramatically alter the switching activity of that function.

Several existing low-power synthesis algorithms address the issue of input switching probabilities other than 0.5 [5]. However, these generally are focused on trying to reduce the functional support from high-activity inputs. One significant problem with this technique is that it supposes that the power optimality of radically different functional representations can be compared prior to logic implementation. The work presented in this paper is a more rigorous formalism for exploiting the distribution of probabilities throughout the Boolean space without forcing support changes. In this way, it builds upon the well-established techniques for area optimization.

The approach to low-power synthesis described here uses a manipulation of the DC set to optimally favour area optimization which also minimizes power. The Boolean space is separated into a set of minterm classes. Each minterm class is defined by the average minterm probability within it. It is shown that there often exists a set of classes with the same total probability of occurrence, but containing exponentially increasing proportions of the functional space. The overlap of the DC set with the classes containing the most minterms can then be used for optimizing the area of a logic function implementation without significantly affecting output probability. The overlap of the DC set with the classes containing the fewest minterms tends to strongly influence the output switching probability during area optimization. The former case is used when a node function already has a low switching probability, the latter is used to favour reduction of switching activity at hot nodes.

The concept of the probability distribution within the Boolean space is described in Sect. 3 after a quick definition of CMOS power consumption (Sect. 2). In Sect. 4 an outline is given for the use of these classes in power optimization, and Sect. 5 is a presentation of the results of applying this theory to standard benchmarks.

2 Power Dissipation in CMOS Logic Circuits

The energy dissipation of a CMOS circuit is directly related to the switching activity when a simplified model of energy dissipation is used. The assumptions in the simplified model are: (1) all capacitance is lumped at the output node of a gate; and (2) current flows only from the supply rail to the load capacitor, or current flows from the load capacitor to the ground rail; and (3) all voltage changes are full swings, i.e. from the supply rail to the ground rail voltage, or vice-versa.

For a well-designed gate, the above assumptions are reasonable [3]. For a synchronous digital system, the average power dissipated by a gate g_i is given by:

$$P_i = \frac{1}{2} \cdot C_i \cdot \frac{V_{dd}^2}{T} \cdot E_i \tag{1}$$

where P_i denotes the average power dissipated by gate g_i, C_i is the load capacitance at the output of gate g_i, V_{dd} is the supply voltage, T is the clock period, and E_i is the average number of gate output transitions per clock cycle. Given a technology-mapped circuit or a circuit layout, all of the parameters in Eqn. 1 can be determined, except for E_i, which depends on both the logic function being performed and the statistical properties of the primary input signals.

Eqn. 1 is used by the power estimation techniques such as [1] [2] [4] to relate switching activity to power dissipation. The theory presented in this paper assumes that the network primary inputs are independent. Furthermore, it is assumed that functional switching power (the zero-delay model) is the predominant effect in determining power consumption. Under this assumption, the power consumption at a gate g_i with onset probability p_i is given by:

$$P_i = C_i \cdot \frac{V_{dd}^2}{T} \cdot p_i(1 - p_i) \tag{2}$$

When load capacitance is constant, functional power minimization is equivalent to optimizing p_i to be close to zero or one.

3 Probability Classes in the Boolean Space

To define the concept of a probability distribution throughout the Boolean space, consider first a simple example. Take the Boolean space described by two variables $\{x, y\}$ where $Pr(x) = 0.2$ and $Pr(y) = 0.4$. There are four minterms which describe this space: $\{\bar{x}\bar{y}, \bar{x}y, x\bar{y}, xy\}$. which have respective probabilities: $\{0.48, 0.32, 0.12, 0.08\}$. For this example there are no two minterms with the same probability of occurrence, and 80% of the total probability is contained by the two minterms within the space described by $f(x, y) = \bar{x}$. Note also that if the space is partitioned according to: $f(x, y) = \bar{x}$ and $f(x, y) = x$, the minterm probabilities in each partition only differ by 0.04 allowing them to be well approximated by their respective averages.

It is desirable to partition the space into sets of minterms all with similar probability. These partitions will be referred to as "classes". This approximate equality in minterm probability within a class implies that functional probability is well approximated by a simple proportionality relationship to the number of onset minterms contained in each class. Furthermore, these classes can be ranked in terms of their likelihood to influence switching activity of a function if used during area optimization.

The definition of suitable classes follows from an analysis of the relationship between functionality and probability. Consider a circuit with four inputs $I_p = \{w, x, y, z\}$, each with probability p of being 1. The distinct minterm probabilities correspond to the number of different ways of choosing n inputs to be on, and $(|I_p| - n)$ inputs off for each $\{n : 0 < |I_p| < n\}$. For example, there are $\binom{4}{2} = 6$ minterms with probability $p^2(1-p)^2$, namely: $\{wx\bar{y}\bar{z}, w\bar{x}y\bar{z}, \bar{w}xy\bar{z}, w\bar{x}\bar{y}z, \bar{w}x\bar{y}z, \bar{w}\bar{x}yz\}$. These sets of minterms form a set of classes $\varphi^0_{I_p}$, the zero superscript indicating that approximating the minterm probability in each class with the class average is exact. By construction, this is a set of $(|I_p| + 1)$ distinct classes. Now consider a circuit with a set of inputs I with onset probabilities P where $|P| \leq |I|$. A set (although not necessarily minimum) of exact-average classes which partition the $|I|$ variable Boolean space is then given by the product of the exact-average classes for each I_p, $p \in P$. i.e. $\varphi^0_I = \prod_{p \in P} \varphi^0_{I_p}$.

Although $|\varphi^0_I|$ is not always minimum, it does show that the number of zero-error classes is $O(\prod_{p \in P}(|I_p| + 1))$, which is generally impractically large even if $|P| << |I|$. (For example, even if $|P| = 5$ and $|I| = 30$, $|\varphi^0_I|$ could be as large as: $(\frac{30}{5} + 1)^5 \simeq 17$ thousand!). Furthermore, as the classes are formed from the "choose" operation, they bear similarities to XOR functions and are in

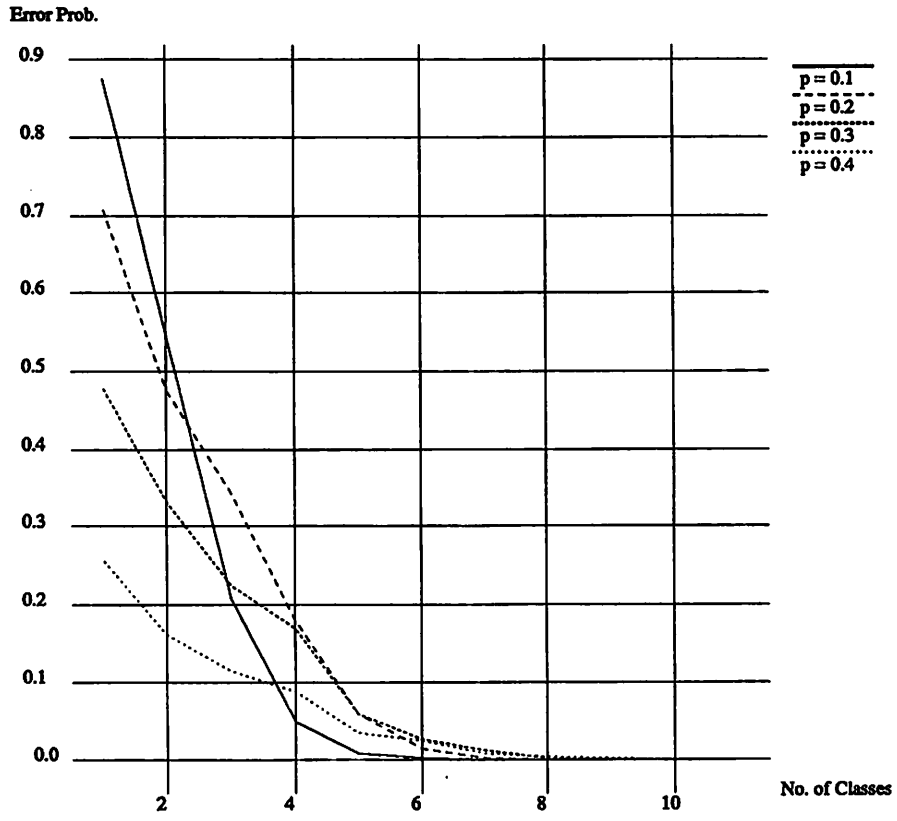

Figure 1: Error vs. Number of Classes: $|I_p| = 10$

some sense very disjoint in the Boolean space. In general, smooth functions which are subsets of the DC set are more suited for synthesis optimization. Consequently, if the intention is to use the classes to modify use of the DC set, it is necessary to deal with smoother functions. A class in φ_I^0 function can be smoothed by collapsing classes within a $\varphi_{I_p}^0$. As was shown in the first example, this collapse can be made without significantly affecting error of approximation by the average class minterm probability. A set of Boolean classes for the entire space with minimal minterm-probability variance can be built from considering error/class count trade-off curves for each I_p.

The maximum error in a class C_i is defined as: $\max_{A \subset C_i} |(Pr(A) - |A|.\frac{Pr(C_i)}{|C_i|})|$. For a specific $p \in P$, let $\varphi_{I_p} = \{C_1, C_2, ...\}$ be a set of classes chosen from the union of classes in $\varphi_{I_p}^0 = \{C_1^0, C_2^0, ...\}$. The maximum error will be minimized if the subset of $\varphi_{I_p}^0$ corresponding to any $C_i \in \varphi_{I_p}$ is contiguous with respect to indicies of $\varphi_{I_p}^0$. eg. $C_1 = C_1^0 \cup C_2^0, C_2 = C_3^0 \cup C_4^0$ will have smaller error than $C_1 = C_1^0 \cup C_3^0, C_2 = C_2^0 \cup C_4^0$. Generation of φ_{I_p} for minimal error then becomes equivalent to the optimal selection of a set of contiguous, mutually exclusive subsets of $\varphi_{I_p}^0$. A heuristic technique to do this is the selection of the subsets of $\varphi_{I_p}^0$ in order to maximize the uniformity of the total probability contained in each of the final classes.

A plot of the total error (which is the sum of the error for each class) in φ_{I_p} against the total number of classes for this grouping strategy is shown in Fig. 1. For each curve, $|I_p| = 10$ and $p \in \{0.1, 0.2, 0.3, 0.4\}$. Note that in each case, the number of classes can be significantly reduced with minimal impact on error. For small allowable error ($< 10\%$), this effect increases with decrease in p.

Now consider the set of classes described by the Boolean product of classes chosen for each input subset I_p. As each set of classes φ_{I_p} covers the entire space with mutually exclusive sets, the overall set of classes formed from the product maintains this necessary property. Let ϵ_p be the total error for the set of classes φ_{I_p}. An upper bound on the error for the set of classes on I defined by the product is given by:

$$\epsilon = 1 - \prod_{p \in P} (1 - \epsilon_p)$$

The tightness of this bound is illustrated in Fig. 2. The data on this graph is generated from a series of statistical tests for input probabilities $p \in \{0.1, 0.2, 0.3, 0.4\}$ with $|I_p|$ chosen randomly from $[2, 10]$ and $|\varphi_{I_p}|$ from $[0, |I_p|]$. It is clear that the error/class count tradeoff curves generated for each I_p can be used to define an error/class count sensitivity for the entire class product φ_I.

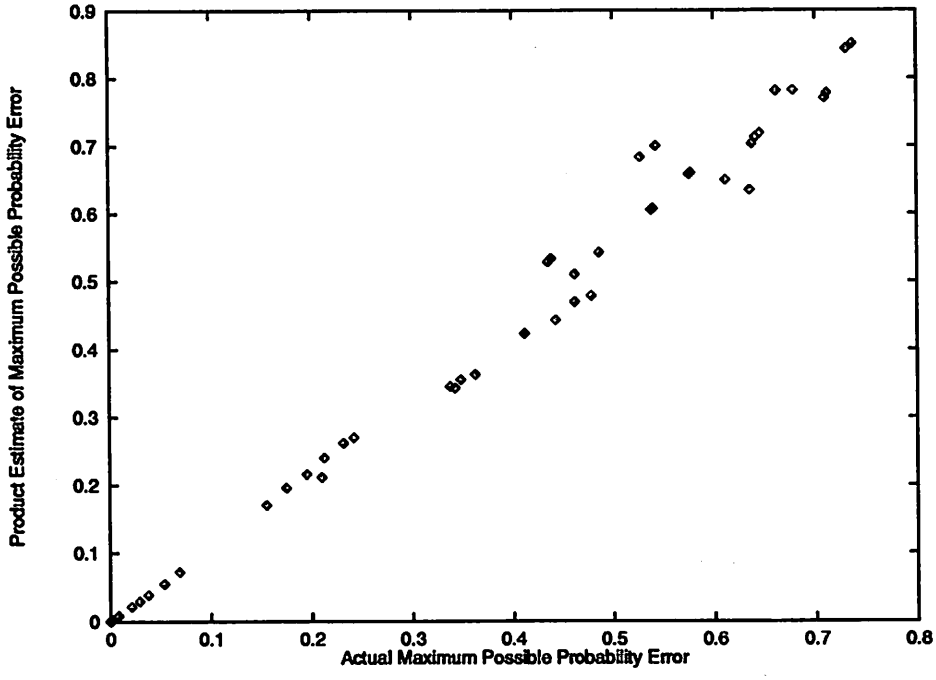

Figure 2: Actual Global Error vs. Product of Errors Estimate

The practical implementation of this work first approximates the entire set of input probabilities (every input has an onset or offset probability in the range of 0 to 0.5) with a set P, $|P| = 5$, chosen

for minimum RMS error. The error curves are then generated for each $p \in P$. The reduction in the number of classes achieved this way with less than a 10% error tolerance can be several orders of magnitude for large circuits.

In general, the number of classes remaining after this operation is still too many for practical use in synthesis. We made the decision that five classes were sufficient for partitioning the space into distinct regions for biasing synthesis. The grouping into the final five classes is based upon the similarity of average minterm probabilities in the original classes, and an attempt to equally partition the total probability.

An example of the distribution of minterms between classes is shown in Fig. 3 for a Boolean space of 30 variables. The curves shown are for five randomly generated sets of variable probabilities. Each of the generated classes correspond to approximately the same total probability, namely 0.2. Note that in each case, over 40% of the total probability is held in about 1% of the Boolean space, and about 80% of the Boolean space falls within 20% of the total probability. The exponential characteristic revealed by these classes is critical to biasing an area optimization tool towards reduced power consumption.

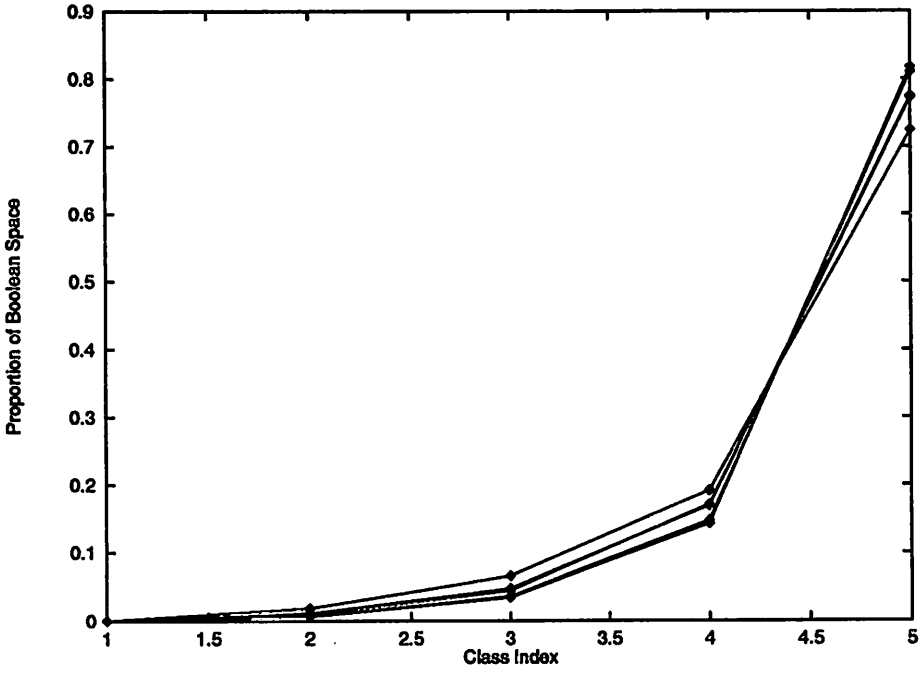

Figure 3: Proportion of Boolean Space in Equi-Probability Classes

4 Biasing Area Optimization Towards Power Reduction

The goal of logic synthesis is generation and optimization of a multi-level logic description which implements a specified function. In this work, the objective is minimization of power and area. Logic optimization makes use of the fact that nodes internal to a network do not generally have a uniquely specified function for satisfying correctness of an implementation. A subset off the Boolean space known as the Don't Care (DC) set can be generated at each node which gives the range of functionality possible during a valid optimization step [6]. These DC sets are usually constructed a single time for the entire network in such a way as to ensure that each node can be optimized independently. This is known as the Compatible DC (CDC) Set construction. Every CDC contains a subset known as the Observability CDC (OCDC) set within which functional manipulation influences node switching probability. (The remainder of the CDC, the Satisfiability CDC, describes conditions on intermediate inputs to a node which cannot be logically satisfied. Consequently, any optimization of functionality within this subset does not influence switching probability.) The contribution of the work presented here is a technique for directing use of the OCDC to maximize reduction in switching activity.

After the Boolean space is split into five classes, subsets of the OCDC are computed to maximize the influence of area optimization upon switching activity. As power optimization is a combination of trying to reduce both area and switching activity, it is important that providing subsets of the DC set to the area optimization algorithm does not reduce available flexibility. This is achieved by two phase process which first addresses reduction in switching activity, then reduction in area.

The concept of separating the optimization phase for reduction in activity and area (therefore, capacitance) is illustrated in Fig. 4. Fig. 4a indicates two classes which partition the Boolean space, each of equal total probability but with C_L encompassing most of the Boolean space as it contains many small probability minterms. Fig. 4b is a node function, f_n, and its ODC set, D_n. Assume that the onset probability for f_n, p_n, is greater than 0.5. To reduce switching activity, we need to increase the onset probability to push it towards 1. Thus it it is desirable during the optimization of f_n to absorb elements of C_H with a functional expansion into the set described by: $\overline{f_n}.f(D_n).f(C_H)$. This region is represented by the black shaded area of Fig. 4c. Clearly, although any expansion will greatly benefit switching activity, this is a very small subset of the DC set and unlikely to provide sufficient area optimality. To optimize for area, flexibility in functional representation is important so a large subset of the DC set must be provided. However, it is important to avoid functional

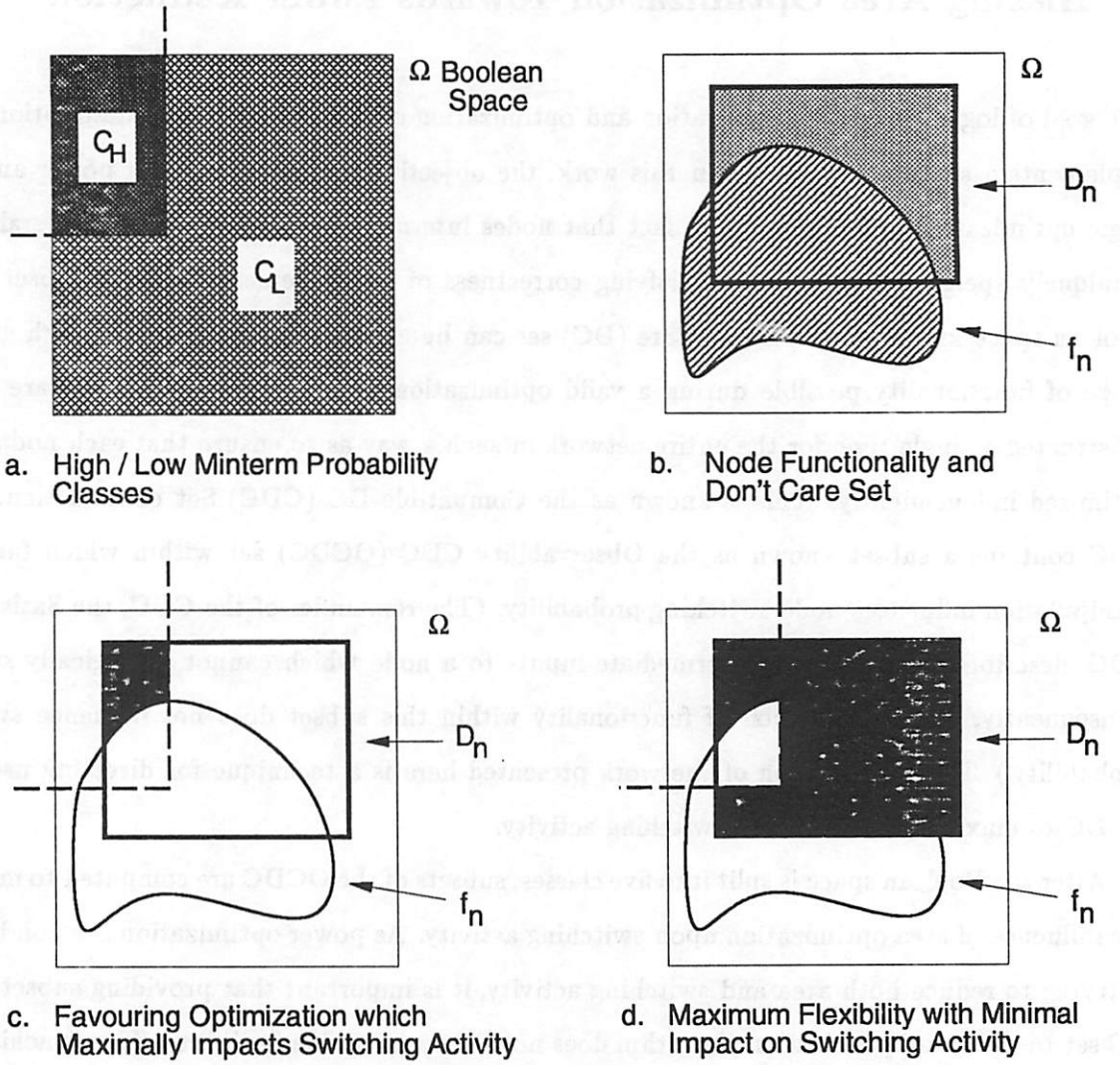

Figure 4: Encouraging Power and Area Optimization as Separate Phases, $p_n > 0.5$

contraction within the set C_H as even a small excursion in that direction could dramatically increase switching activity. (i.e. p_n would be move significantly closer to 0.5). The set providing maximum flexibility without allowing small functional changes to strongly influence switching activity is: $f(D_n).(\overline{f_n} + f_n.f(C_H))$. This region is represented by the black shaded area of Fig. 4d. Although the subset of the DC set which is very beneficial for activity optimization is within this set, the disproportionate flexibility provided by the DC set within C_L will tend to reduce the probability that this area optimization alone strongly benefits activity reduction. An algorithm for exploiting the beneficial properties of both these DC set restrictions is now described.

Let $\{C_i : 1 < i < 5\}$ be the set of classes ordered from high to low average minterm probability in the increasing order of class index (as in the example of Fig. 3). Let $f(C_i)$ be the function describing

class i. Consider a node n with function f_n and local CDC set D_n (described by function $f(D_n)$). Let $\mathcal{A}(f_n, f(D_n))$ be the functional manipulation performed on f_n during area optimization within the DC set.

Consider the case of $p(f_n) > 0.5$. To bias area optimization towards reducing switching activity, the onset probability should be increased. The first step is to use the DC set flexibility within the set C_1 (the class with the minterms of largest probability) and the offset of f_n. This will produce a new function f_n' which should have lower switching activity. However, this step provided very little flexibility for area optimization. This is followed by optimizing f_n' with greater flexibility in the set described by: $\overline{f_n'}.f(D_n).(f(C_1) + f(C_2))$. This expansion continues until power is no longer reduced at which point the node function, $f_n(p)$, is said to be activity optimized.

The algorithm then focuses on area optimization by allowing the function probability to shrink. This step is taken in the hope that the flexibility provided by supplying more of the DC set to the optimization strategy will provide a win in power through reduction in capacitance. Consequently, the area optimization step begins by expanding the utilized portion of the DC set *inside* $f_n(p)$. The first expansion provides the most flexibility but the least probability of changing the switching activity in a detrimental way. i.e. $f(D_n).(\overline{f_n(p)} + f(C_5))$. If power is reduced, the $f_n'(p)$ produced by that area optimization is provided further functional flexibility with the set: $f(D_n).(\overline{f_n'(p)} + f(C_5) + f(C_4)$. This expansion across the classes continues until power is no longer reduced.

The procedure is similar if $p(f_n) < 0.5$, but the DC set are initially ANDed with the onset of the function to bias area optimization toward reducing the onset probability.

```
for i = 1 to N
    f_i' = f_i
    if p(f_i) > 0.5
        f_Dopt = NULL
        for j = 1 to 5        /* Low Flexibility, High Expected Activity Change */
            if j = 1 or P(f_i') < P(f_i)
                f_i = f_i'
                f_Dopt = f_Dopt + f(D_n).f(C_j)
                f_i' = A(f_i, (f_i'.f_Dopt))
        f_Dopt = NULL
        for j = 5 to 1        /* High Flexibility, Low Expected Activity Change */
            if j = 1 or P(f_i') < P(f_i)
                f_i = f_i'
                f_Dopt = f_Dopt + f(D_n).f(C_j)
                f_i' = A(f_i, (f_i.f_Dopt + f_i'.f_D_n))
    else
        f_Dopt = NULL
        for j = 1 to 5        /* Low Flexibility, High Expected Activity Change */
            if j = 1 or P(f_i') < P(f_i)
                f_i = f_i'
                f_Dopt = f_Dopt + f(D_n).f(C_j)
                f_i' = A(f_i, (f_i.f_Dopt))
        f_Dopt = NULL
        for j = 5 to 1        /* High Flexibility, Low Expected Activity Change */
            if j = 1 or P(f_i') < P(f_i)
                f_i = f_i'
                f_Dopt = f_Dopt + f(D_n).f(C_j)
                f_i' = A(f_i, (f_i'.f_Dopt + f_i.f_D_n))
```

Figure 5: Pseudo-code for Favouring Power Reduction during Area Optimization.

A pseudo-code form of this algorithm is provided in Fig. 5. For that presentation, let N be the number of nodes in the network, and $\{n_i : 1 < i < N\}$ be the set of nodes indexed in reverse topological order from the primary outputs to the primary inputs. Assume that the classes and CDCs for the network have already been produced. Let $P(f)$ be the power consumption of the

logical representation of f.

In practice, we found that it was not necessary to test many of the increased flexibility expansions to find optimal solutions. To reduce computation time only two test optimizations are performed at each node; one targeting switching activity and the other area. This had no statistically significant influence upon the power/area optimality of the final result. Biasing optimization towards altering the switching activity for a node involved use of the DC set contained in the two highest minterm probability classes. Optimization for area avoided elements of those same classes which might detrimentally affect switching activity.

5 Results

The algorithms outlined in this paper were implemented inside the SIS logic synthesis package to guide the node minimization phase during multi-level logic optimization. The resulting programs for power-sensitive node minimization - *power_simplify()* and *power_full_simplify()* are counterparts of the area optimization routines *simplify()* and *full_simplify()* of SIS. For benchmarking purposes, we replaced the occurrences of *simplify()* and *full_simplify()* in *script.rugged* with our *power_simplify()* and *power_full_simplify()* command to obtain *script.power*. A subset of the MCNC benchmark set was used to obtain preliminary experimental results. All circuits were mapped using **msu.genlib**. Power estimation and switching activity computation was performed using the symbolic simulation method of [2] using a zero-delay model. All experiments were run on a DEC-station ALPHA with a 160Mb memory.

The results comparing the area and power reduction obtained by optimization via *script.rugged* and *script.power* are presented in Table 1. Column 1 gives the number of literals in the factored form. Column 2 shows the area of the circuit prior to optimization. Columns 3 and 4 present the results after network optimization using *script.rugged* and *script.power* respectively. Similarly, Column 5 shows the power dissipation of the unoptimized circuit and Columns 6 and 7 present the power dissipation results after optimization using *script.rugged* and *script.power* respectively. Columns 8 and 9 show the % change in area and power results by using *script.power* instead of *script.rugged.*

Table 1. Area and Power comparison of *script.rugged* and *script.power*

Circuit	#lit	Area			Power Dissipation (μW)			% change	
		Initial	*s.rugged*	*s.power*	Initial	*s.rugged*	*s.power*	Area	Power
cm82a	31	368	352	360	74.4	91.1	71.1	2.3	-28.1
cm138a	39	480	472	472	67.6	49.2	49.2	-	-
cm42a	39	536	528	456	37.9	59.4	34.3	-13.6	-42.3
cm85a	69	808	824	824	100.0	99.6	99.6	-	-
pm1	65	1000	792	816	198.4	90.3	107.4	3.0	18.9
decod	71	744	704	704	175.2	66.7	66.7	-	-
sct	194	2128	1336	1264	540.1	248.6	210.1	-5.4	-15.5
f51m	207	2272	1840	1384	474.6	356.3	222.9	-24.8	-37.5
lal	252	3856	1616	1616	630.0	312.1	301.1	-	-3.5
9symml	328	3552	3464	3736	912.7	820.4	708.7	7.9	-13.6
ttt2	420	4624	3240	3752	1085.4	458.1	646.2	15.8	41.1
alu2	635	7264	5960	5608	1249.7	931.6	820.5	-5.9	-13.5
vda	1870	19672	10400	8176	3097.1	955.0	803.0	-21.4	-15.4
alu4	1184	13656	11736	10976	3010.8	2223.9	1970.3	-6.5	11.4
Total		60960	43264	40144	8643.1	6762.3	6111.1	-7.2	-9.6

The results demonstrate that in most cases *script.power* yields a circuit with lower power dissipation than *script.rugged*. There is no significant trade-off in terms of circuit area. Infact, on the whole, *script.power* performs better in area optimization as well. Note that *script.power* does not always give a better result than *script.rugged*. This is to be expected since our approach attempts to favorably bias (from the power perspective) the network optimization process at the node minimization level, but cannot guarantee that this will always translate in a lower power network. At the same time, since the area optimization flexibility is not strongly affected by our algorithm, we do expect that in most cases our algorithm will yield a lower power network without any area penalty. This assertion is validated by the experimental results (an average 9.6% reduction in power and a 7.2% reduction in area over the set of benchmark circuits).

We are currently working on testing our approach on other circuits in the ISCAS benchmark set.

6 Conclusions

We have presented a technique to guide area optimization using power-sensitive don't care sets towards low-power implementations without trading-off the synthesis flexibility required for area optimization. The main contributiuons of this work are :

- An approach to formally exploit the large variance in minterm probabilities in the boolean space. We observe that a large part of the probability space is contained in a fraction of the minterms of the boolean space. This can be used to increase/decrease the onset probabilities as required without significantly compromising on the size of the don't care set available for area optimization.
- We present an effective scheme to partition the boolean space in classes of equal probability minterms. This problem is non-trivial in general as the minterm-class construction can be exponential in the number of distinct probabilities at the inputs.
- A heuristic algorithm to direct the synthesis algorithm towards using beneficial minterm classes and avoiding classes detrimental to power dissipation, without making any major reduction in the area optimization flexibility was presented. Experiemental results indicate that this approach can result in lower power implementations of a circuit without incurring any penalty in area.

References

[1] R. Burch, F. Najm, P. Yang, and T. Trick. "McPOWER: A Monte Carlo Approach to Power Estimation." In *Proceedings of the Int'l Conference on Computer-Aided Design*, pp. 90–97, Nov. 1992.

[2] A. Ghosh, S. Devadas, K. Keutzer, and J. White. "Estimation of Average Switching Activity in Combinational and Sequential Circuits." In *Proceedings of the* 29^{th} *Design Automation Conference*, pp. 253–259, June 1992.

[3] L. Glasser and D. Dobberpuhl. *"The Design and Analysis of VLSI Circuits."* Addison-Wesley, 1985.

[4] F. Najm. "Transition Density, A Stochastic Measure of Activity in Digital Circuits." In *Proceedings of the* 28^{th} *Design Automation Conference*, pp. 644–649, June 1991.

[5] S. Iman, M. Pedram. "Multi-Level Network Optimization for Low Power." In *Proceedings of the Int'l Conference on Computer-Aided Design*, pp. 372–377, Nov. 1994.

[6] H. Savoj, R. K. Brayton. "The Use of Observability and External Don't Cares for the Simplification of Multi-Level Networks." in *Proceedings of the* 28^{th} *Design Automation Conference*, pp. 297–301, June 1990.