Computing Netword DCs and Function Minimization Targeting Low Power

Sasan Iman, and Massoud Pedram

CENG Technical Report 95-28

Department of Electrical Engineering - Systems
University of Southern
Los Angeles, California 90089-2562
(213)740-4458

December 1995

Computing Network DCs and Function Minimization Targeting Low Power

Sasan Iman and Massoud Pedram

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, California

Abstract

This paper shows that using don't cares computed for area optimization during local node minimization may result in an increase in the power consumption of other nodes in the network. It then presents techniques for computing Power Relevant Don't Cares that can be freely used to optimize the local function of nodes. These don't cares consist of observability and satisfiability don't care conditions that guarantee that local changes do not increase the power consumption of other nodes in the network and allow the implementation of the node function using inputs with lower switching activities. The concept of minimal variable support is then used to optimize the local function of each node for minimum power using its power relevant don't care set, that is, to re-implement the local function using a modified support that has a lower switching activity. Efficient techniques based on reduced off-sets are also presented for computing the set of all minimal variable supports of a function. Empirical results on a set of benchmark circuits are presented and discussed.

Keywords: Power Relevant Don't Care, Node Power Optimization, Don't Care computation, Minimal Variable Supports, Load Estimation.

1 Introduction

The requirement of portability for new digital applications places severe restrictions on size and power consumption of these units. These new applications often require real time processing capabilities and thus demand high throughput. At the same time, with reductions in the minimum feature size of VLSI designs, power consumption is becoming the limiting factor in the amount of functionality that can be placed on a single chip. Exploring the trade-off between area, performance and power during synthesis and design is thus demanding more attention.

Low power VLSI design can be achieved at various levels of abstraction. For example, at the system level, inactive hardware modules may be automatically turned off to save power. At the architectural level, concurrency increasing and critical path reducing transformations may be used to allow reduction in supply voltage without degrading system throughput [6]. At the device level, threshold voltage of MOS transistors can be reduced to match the reduced supply voltage [15]; Very low threshold voltages may be made possible by electrically controlling the threshold values (against process and temperature variations) by substrate modulation [4].

Once these system level, architectural and technological choices are made, it is the switched capacitance of the logic (switching activity of the logic weighted by the capacitive loading) that determines the power consumption of a circuit. Logic synthesis has proved to be an effective part of any design cycle by effectively changing the circuit structure to optimize area and delay. New approaches can also be used at the logic synthesis level to minimize the switched capacitance of the circuit.

Function minimization using a don't care set is an important part of logic synthesis for minimum area. The problem of don't care computations and Boolean function minimization have been addressed by many researchers in the past. ESPRESSO [3] presents a heuristic approach where novel techniques are used to efficiently produce good area solutions while ESPRESSO EXACT [20] presents an exact method for solving the minimum area solution. In [8] a method is presented for computing the complete set of observability conditions for each node in the network where the observability don't care (*ODC*) at each node is computed as a function of the *ODC* for its fanout edges.

In this paper we address the problem of Boolean function minimization for low power using a don't care set. The problem of power optimization during logic synthesis has been addressed in a number of recent publications. The don't care set computed for area optimization [21] is used in [22] to optimize the local function of nodes for power. This work does not however take into account the effect of changes in the function of internal nodes on the power consumption of other nodes in the network. The idea of power relevant don't cares was first introduced in [12] where an efficient technique for computing power relevant don't cares was presented. The computed power relevant don't care guarantees that any changes in the local function of a node does not result in increasing the switching activity of other nodes in the network beyond their value *when these nodes were optimized*. The notion of minimal variable supports is used in [12] to optimize the local function of nodes for power.

The analysis on power relevant don't cares [12] is used in [14] to compute a re-synthesis potential for nodes in a technology mapped network. This re-synthesis potential represents the estimated effect of a change in the local function of a node on the power consumption of its transitive fanout nodes. The method presented in [14] also takes into account changes in power consumption due to variations in hazardous transitions in the network after an internal node is re-synthesized. Under a simplifying assumption (that is, assuming a signal probability of 0.5 for all primary inputs) an efficient technique is presented for computing this re-synthesis potential.

This first part of this paper presents a method for computing the power relevant observability and satisfiability don't cares for nodes in a Boolean network. The power relevant observability don't

care presented here consists of the power relevant don't cares presented in [12]. In addition, an extension to this theory is presented that makes possible computation of Monotone Power Relevant Observability Don't Cares. Using this don't care set, it is guaranteed that local node optimization do not increase the switching activity of other nodes *beyond their current value*. This results in a monotone decrease in the switching activity of the network as nodes are optimized. The power relevant satisfiability don't care conditions included in the don't care set, are intended to maximize the probability of substituting nodes with low switching activity into the node being optimized. The approach to compute the satisfiability don't care [21] is modified in this paper to include low switching activity nodes (from which a subset of nodes can be selected for possible substitution into the function of node being optimized).

The second part of the paper uses the notion of minimal variable supports to find a minimal power implementation of the node function. Using this approach and also using the power relevant satisfiability don't cares described above, inputs with high switching activity may be replaced with new fanins with lower switching activities when permitted by the power relevant don't care conditions derived from the network.

The rest of this paper is organized as follows. In section 2 we present switching activity, load and power estimation models. In section 3 we present techniques for computing the set of power relevant don't cares. We then use power relevant don't cares in section 4 to present a method for optimizing the local function of nodes for low power using the concept of minimal variable supports. Results and conclusion are presented in sections 5 and 6.

2 Power Model

2.1 Computing Switching Activities

We use the following method to compute the switching activity for all nodes in the network for purposes of power estimation and optimization. We use the signal probabilities at the primary inputs to compute the signal probability for each internal node by building its global BDD. We then compute the switching activity of the node assuming temporal independence at the primary inputs. Given the signal probability p_n for an internal node n, the switching activity for n is computed using the following equation:

$$E(n) = 2 \cdot p_n \cdot (1 - p_n) \tag{1}$$

This method for estimating switching activity ignores the temporal and spatial correlations[18] at the primary inputs of the circuit. It does however account for spatial dependence due to reconvergent fanout paths inside the network.

The analysis presented in the second part of this paper for optimizing the local function of nodes does not make any assumption on how the switching activities are estimated. In fact this procedure uses the switching activity values to guide the optimization process. Therefore more accurate switching activity estimation techniques that also take into account spatiotemporal correlations can easily be used during the node optimization process.

The analysis presented in this paper for computing power relevant don't care conditions is carried out assuming that the switching activities are estimated using equation 1. The goal here is to compute don't care conditions that result in minimizing switching activities estimated using this model. Minimizing the switching activity in this case is equivalent to changing the signal probability such that the signal probability is very close to 0 or 1 which is the same as changing the node function such that most of the time it evaluates to either logic value zero or logic value one. This is equivalent to attempting to "turn off" the gates in the network when permitted by their observability conditions. In other words, using power relevant observability conditions is similar to "turning off" parts of the logic which are not being used for computing the output function as presented in [1] and [23].

2.2 Load estimation

Previous methods for power estimation have used number of fanouts for a node as an estimate of the load at the output of a node. In this paper we use "load in the factored form" to compute the load at the output of a node. Assume that i is the variable associated with the fanout edge of node n_i in the Boolean network. Then the factored load of node n_i is the given by

$$L_i = \sum_{n_k \in fanouts} FL(n_i, n_k) \tag{2}$$

where $FL(n_i, n_k)$ gives the number of times variable i is used in the factored form of the function associated with fanout node n_k.

The following example shown in figure 1 illustrates this computation.

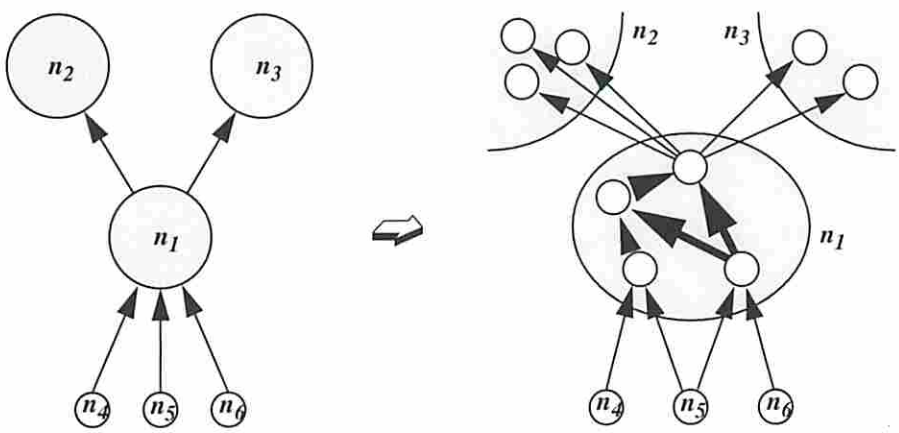

Figure 1: Load in the factored form

Example 1:

In order to use the new method we first observe the following:

$FL(4,1) = 1$ $FL(5,1) = 2$

$FL(6,1) = 1$ $FL(1,2) = 3$

$FL(1,3) = 2$

Then

$L_1 = 5$ $L_5 = 2$

$L_4 = 1$ $L_6 = 1$

■

Experimental results on the accuracy of load estimation techniques show load in the factored form of nodes in the technology independent network to be an accurate measure of load for the gates in the technology mapped network [11].

2.3 Power Model for Node Functions

Using the definition for factored load we define the factored form power cost function for a node. Given a node n_i, the power cost of the node in the factored form is computed as:

$$P^{FAC}_{n_i} = E(f_i) \cdot \sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) + \sum_{n_m \in Internal(n_i)} E(n_m) + \sum_{n_j \in fanins(n_i)} E(n_j) \cdot FL(n_j, n_i) \quad (3)$$

where $Internal(n_i)$ is the set of all internal nodes in the factored form representation of n_i and f_i is the function of node n_i.

Note that in equation 3, load for internal nodes is assumed to be 1. This is a valid assumption since procedures for factorizing a node [24] and their extension for power consideration [13] guarantee that each internal node has only one fanout.

3 Power Relevant Don't Cares

Logic synthesis algorithms use the flexibility provided by don't care conditions to more effectively manipulate a Boolean network. Current techniques for computing and using these don't care conditions allow for correct functional operation of the network by guaranteeing that as each function in the network is being optimized using its don't care conditions, other functions in the network are changed only within their don't care sets.

Power consumption in a Boolean network is proportional to the switching activity of the nodes in the network. Switching activity of a node is a function of the signal probability of the function and therefore dependent on the global function of the node. This means that as a node is being optimized using its don't care set, the switching activity of the node function as well as the switching activity of other nodes in the network is also being changed. This clearly shows the need for new methods to analyze the effect of using don't care sets on the switching activity of nodes in the network. This section provides such an analysis. The analysis presented in this section is used to propose new techniques for computing and using power relevant observability don't care sets. Power relevant satisfiability don't cares are the subject of section 3.6.

3.1 Don't Care Conditions for Area

In logic synthesis, the concept of don't cares is used to represent the available flexibility in implementing Boolean functions. Don't care conditions for a function specify part of the Boolean space where the function can evaluate to one or zero. Three sources of don't cares are external don't cares (*EDC*), observability don't cares (*ODC*) and satisfiability don't care(*SDC*). It has been shown that if a node is minimized using all three types of don't cares, then all connections to and inside the node are irredundant. This means that *EDC*, *ODC* and *SDC* provide a complete set of don't care during optimization [2].

Definition 1 *The external don't care set for each output z_i of the network is all input combinations that either do not occur or the value of z_i for that input combination is not important.*

Definition 2 *If y_i is the variable at an intermediate node and f_i its logic function, then $y_i = f_i$. Therefore, we don't care if $y_i \neq f_i$. The expression $SDC = \sum (y_i \oplus f_i)$ for all nodes in the network is called the Satisfiability Don't Care set (SDC).*

Definition 3 *The observability don't cares (ODCs) at each intermediate node y_0 of a multi-level network are conditions under which y_0 can be either 1 or 0 while the functions generated at each primary output remain unchanged. If $z = (z_1, \ldots, z_l)$ gives the set of circuit outputs, then the complete ODC at node y_0 is:*

$$ODC_0 = \langle m \in B^n | z_{y_0}(m) \equiv z_{\tilde{y}_0}(m) \rangle = \prod_{i=1}^{l} \overline{\frac{\partial z_i}{\partial y_0}} \quad (4)$$

3.1.1 Computing Don't Care Conditions

The complete don't care set for a node n is found by first computing the *ODC* as a function of the primary inputs of the circuit. The external don't care which is also expressed in terms of the primary inputs is then added to *ODC*. Image projection techniques [7] are then used to find the *ODC* plus *EDC* of the node in terms of the immediate fanins of the node. Finally a subset of *SDC* for nodes which can be substituted into n with high probability, is added to this local don't care. In general computing the *ODC* for a node is the most complex part of this computation. In the following we briefly describe this procedure.

In [8] a method is described for computing the complete set of observability conditions for each node in the network where the *ODC* at each node is computed as a function of the *ODC* for its fanout edges. In this procedure the *ODC* of the node with respect to each primary output is computed separately. A different technique for computing the complete don't care set is presented in [21] which takes advantage of observability relations [5] at the primary outputs of the network. The given algorithm computes the complete *ODC* for each node in a multi-level combinational network. For tree networks, the following equation is used to compute the maximal set of *ODC*s at the output a node g.

$$ODC_g = ODC_f + ODC_g^f \tag{5}$$

where $ODC_g^f = \overline{\frac{\partial f}{\partial g}}$.

The complete *ODC* cannot however be used in synthesis for any real size problem. This is because once the function of a node is minimized using its complete *ODC*, the *ODC* at other nodes in the network will potentially change. Therefore the *ODC* for each node has to be recalculated after each optimization step. At the same time, the size of the complete *ODC* can become extremely large. Therefore subsets of *ODC* have to be used instead of the complete *ODC* of each node. *RESTRICT* was the first *ODC* filter introduced in [10]. This filter removed any cube in the *ODC* of a node y_i which had a literal corresponding to a node in its transitive fanout. Although this filter and a number of other filters made the *ODC*s smaller, *ODC*s still had to be recomputed after each node simplification. Compatible set of permissible functions (*CSPF*), introduced in [19] allows for simultaneous optimization of all nodes in a network. In [21] the concept of *CSPFs* which is only defined for NOR gates, is extended to complex nodes of a general multi-level network and is called Compatible *ODC*s (*CODCs*). *CODCs* are used to simultaneously minimize the function of each node in the network. Even though by using *CODCs* some of the information contained in each of the full *ODCs* is lost, the time complexity of using the full *ODC* makes it impossible to use them for any practical size problem. The following equation presented in [21] is used for calculating the maximal set of *CODC* for the fanins of a node g in a tree network.

$$CODC_g = CODC_g^f + CODC_f \tag{6}$$

Two method are also presented for computing $CODC^f_g$. The first method produces the maximal set for $CODC^f_g$ but is shown to be too expensive. The second method shown by the following equation is a more efficient technique for computing a valid subset of the maximal set of $CODC^f_g$.

$$CODC_g^f = \left(\frac{\partial f}{\partial y_1} + C_{y_1}\right).....\left(\frac{\partial f}{\partial y_{k-1}} + C_{y_{k-1}}\right)\overline{\frac{\partial f}{\partial y_k}} \tag{7}$$

In this equation g is assumed to be the *kth* input y_k to function f and also the variable ordering $y_1>....>y_n$ is assumed for the fanins of function f. Also, $C_x(f) = f_x \cdot f_{\bar{x}}$ is the largest function independent of variable x which is contained in function f.

These results are then generalized to multiple fanout nodes by finding the intersection of the compatible *ODCs* for each fanout edge of a node as shown in equation 8,

$$CODC_g = \prod_{f_i \in fanouts} CODC_g(f_i) \quad (8)$$

where $CODC_g(f_i)$ gives the compatible *ODC* computed using equation 6 for fanout f_i of node g.

This method for computing the compatible don't care set requires that each node is optimized only after all its fanout nodes have been optimized. The procedure shown in figure 2 is used to optimize the function of nodes in the network using the computed don't care set.

```
1:  function full_simplify(G)
2:  G(V, E) is a Boolean network;
3:  begin
4:      for node ∈ G in reverse dfs order do
5:          CODC  = computeCODC(node)
6:          CODC' = computeLocalDC(node, CODC)
7:          SDC = selectNodeSDC(node)
8:          newf = nodeSimplify(node, CODC', SDC)
9:      endfor
10: end
```

Figure 2: Computing compatible *ODCs* in a network

Function *computeCODC* uses equation 8 to compute the compatible function in terms of the primary inputs. Function *computeLocalDC* will then find the don't care in terms of the immediate fanins of the function using image projection techniques. The function of the node is then optimized using the local *CODC* and a subset of the SDC for nodes which with high probability may be substituted into the node.

3.2 Observability Don't Care Conditions for Power

The compatible don't care computed in section 3.1 is freely used while minimizing the function of nodes in a Boolean network guaranteeing that the global function of circuit outputs will only change within their specified external don't care set. The change in the global function of transitive fanouts of node n as n is optimized, is not of concern when area is being minimized since the change in the function of each node will be within the observability don't care calculated for that node. However as mentioned before, this observation does not hold for power minimization. For example if by modifying the function of an internal node, the signal probability of a fanout node is changed from 0.1 to 0.2 we can expect 78% increase in the power consumption of the fanout node. The following example demonstrates how optimizing the function of an internal node for power may result in an increase in the total power consumption of a network.

Example 2: Consider a function f with a load of 5 and function g with load 1 where:

$f = g + a$;

$g = \bar{a} \cdot b$;

Also: $dc_f = \bar{a}.b$, *also* $p(a)=p(b)=0.5$

Then $DC_g = dc_f + a = a + b$

Before optimization:

$f = a + \bar{a}.b \Rightarrow p(f) = 3/4 \Rightarrow E(f) = 3/8 \Rightarrow Power(f) = 15/8$

$g = \bar{a}.b \Rightarrow p(g) = 1/4 \Rightarrow E(f) = 3/8 \Rightarrow Power(g) = 3/8$

After optimization, function g is set to 0:

$f = a \Rightarrow p(f) = 2/4 \Rightarrow E(f) = 4/8 \Rightarrow Power(f) = 20/8$

$g = 0 \Rightarrow p(g) = 0 \Rightarrow E(f) = 0 \Rightarrow Power(g) = 0$

This example shows that after optimizing node g, the power at the output of node f is increased.

3.3 Observability Don't Care Analysis for Low Power Node Minimization

In this section we present an analysis on the effect of using observability don't cares on the signal probability of other nodes in a tree network. This analysis will then be used to derive exact and heuristic methods for computing power relevant observability don't cares.

3.3.1 Observability Don't Care Analysis for Tree Networks

Assume a function f and its fanin g. Figure 3 shows the relationship between the global function g and its maximal *ODC* set. The space of the primary inputs is shown by the points inside large circle. Points inside the smaller circle represent the on-set points for function g. The single dashed region gives all conditions where changes in function g is unobservable at node f. The combination of the single and cross dashed regions give all the primary input combinations where changes in g is unobservable at the primary outputs. Note that $g \cap ODC_g \neq \emptyset$.

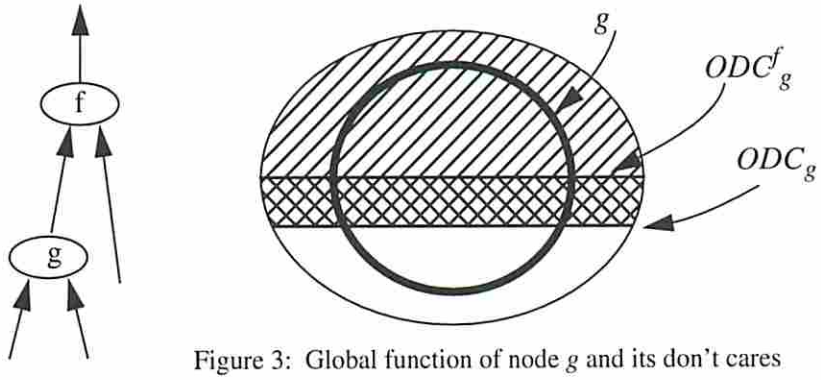

Figure 3: Global function of node g and its don't cares

The following lemmas are used to study the effect of changes in the function of node g in the function f.

Lemma 1 *$\partial f^+/\partial g$ as defined here gives all global conditions where both functions f and g evaluate to the same value.*

$$\frac{\partial f^+}{\partial g} = f_g \cdot \overline{f_{\bar{g}}} \tag{9}$$

Proof: The following relations exists between f and g where f_g is the cofactor of f with respect to variable g:

f_g: global conditions where $f = 1$ if $g = 1$

$\overline{f_{\bar{g}}}$: global conditions where $f = 0$ if $g = 0$

The intersection of these two functions gives all conditions where f changes from 0 to 1 when g changes from 0 to 1. In other words:

$\{ \partial f^+/\partial g \subseteq B^n \mid \forall v \in B^n, f(v) = g(v) \}$.

■

Lemma 2 *$\partial f^-/\partial g$ as defined here gives all global conditions where functions f and g evaluate to opposite values.*

$$\frac{\partial f^-}{\partial g} = \overline{f_g} \cdot f_{\bar{g}} \tag{10}$$

Proof: The following relations exists between f and g:

$f_{\bar{g}}$: global conditions where f =1 if $g = 0$

f_{g}: global conditions where f =0 if $g = 1$

The intersection of these two functions gives all conditions where f changes from *1* to *0* when g changes from *0* to *1*. In other words:

$\{ \partial f^{-}/\partial g \subseteq B^n \mid \forall v \in B^n, f(v) = g(v)\}$.

■

Note that $\partial f/\partial g = \partial f^{+}/\partial g + \partial f^{-}/\partial g$ which is the difference equation and also $\partial f^{+}/\partial g \cap \partial f^{-}/\partial g = \emptyset$.

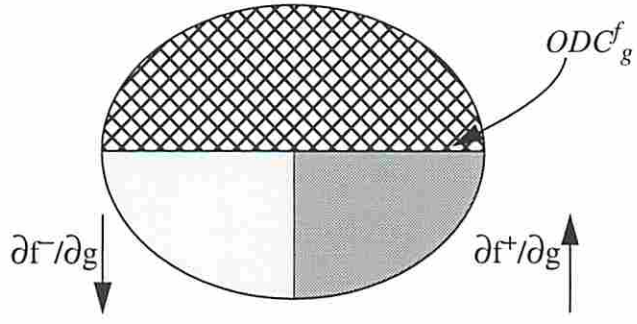

Figure 4: Partition of the space of primary inputs for function f and g

Figure 4 shows how the global space of the primary inputs is partitioned with respect to global functions of f and g. The region specified by ODC^f_g specifies all points of the global space where changes in g will not affect the global function of f. Region $\partial f^{+}/\partial g$ specifies all points in the global space which if included in the on-set of g will also be included in the on-set of f. This means that by including these points in g, the number of minterms in f will also increase (hence the up arrow). Region $\partial f^{-}/\partial g$ specifies all the points in the global space which if included in the on-set of g will be removed from the on-set of f. This means that including these points in g will reduce the number of points in the onset of f (hence the down arrow).

3.3.2 Observability Don't Care Regions

Figure 5 shows the relationship between the global functions of $\partial f^{-}/\partial g$, $\partial f^{+}/\partial g$ and g. This figure is obtained by overlapping figures 3 and 4. Even though no assumption is made about the global function of nodes f and g, figure *5* shows all possible combinations of the regions shown in figures 3 and 4 assuming a tree network.

Figure 5: goes here.

The global don't care conditions for node g (region above line ODC_g in figure *5*) is partitioned into six regions. Each of these regions specifies a don't care region of g with respect to f as defined in the following.

Definition 4 *Given a node g and its fanout node f, the don't care regions of g with respect to f are denoted as* $R_{g,f}(\alpha, \beta)$ *. This don't care region specifies all global conditions where g evaluates to* α

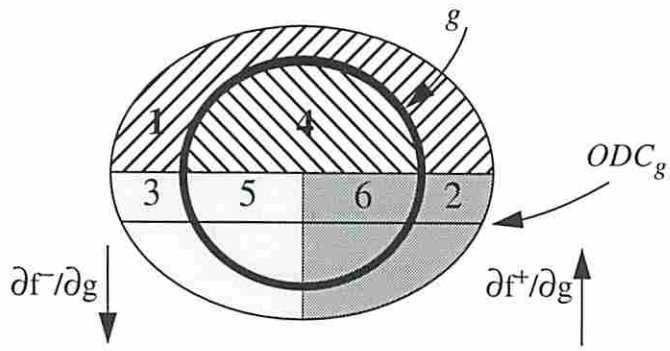

Figure 5: Don't care regions for functions f and g

(α={0,1}). The second entry, β={0,1,-} specifies whether for points in this region f evaluates to the same value as g (β=0), the opposite value of g (β=1) or whether f is independent of g (β=-).

For example region 3 in figure 5 is denoted as $R_{g,f}(0,1)$ and region 4 is given as $R_{g,f}(1,-)$.The definition for don't care regions is extended in the following to represent the relationship between a node g and its transitive fanouts nodes.

Definition 5 *Given a node g and its fanout nodes $F=\{f_1,...f_k\}$, the don't care regions of g with respect to F are denoted as $R_{g,F}(\alpha, \beta)$. This don't care region specifies all global conditions where g evaluates to α (α={0,1}). The second entry is a k bit vector where each bit takes values from the set β={0,1,-}. Bit i of this vector specifies whether for points in this region f_i evaluates to the same value as g (β_i=0), an opposite value than g (β_i=1) or whether f_i is independent of g (β_i=-).*

The following lemmas give properties of don't care regions which will be used to study the effect of changing the global function of node g on the global function of its fanout node f.

Lemma 3 *Using the minterms in $R_{g,f}(1,-)$ and $R_{g,f}(0,-)$ while optimizing node g will not affect the global function of node f.*

Proof: The proof follows immediately from the definition for don't care regions.

■

Lemma 4 *Using a minterm v_i in don't care regions $R_{g,f}(1,1)$ and $R_{g,f}(0,0)$ while optimizing node g will result in bringing this minterm from the off-set to the on-set of function f.*

Proof: For all minterms in region $R_{g,f}(0,0)$ ($R_{g,f}(1,1)$) function g evaluates to $0(1)$. This means that using a minterm in this region while optimizing function g is equivalent to including this minterm in the on-set (off-set) of node g. Also for all minterms in this region, f and g evaluate to same(opposite) values. This means that for all minterms in this region function f evaluates to 0. Therefore including a minterm in this region in the on-set (off-set) of g will also include this minterm in the on-set of f.

■

Lemma 5 *Using a minterm v_i in don't care regions $R_{g,f}(0,1)$ and $R_{g,f}(1,0)$ while optimizing node g will result in bringing this minterm from the on-set to the off-set of function f.*

Proof: proof is similar to the previous lemma.

■

Table 1 summarizes the results of these lemmas. In this table, v is a minterm in the space of the primary inputs of the network.

For each region in ODC_g, the change in the function of f as minterm v_i in region i is included in or excluded from the on-set of g is well defined. This means that while optimizing node g, the effect of changes in global function of node f is exactly known using the information on the don't care regions for node g. Therefore the effect of changing the function of g on the signal probability and therefore the switching activity of node f can exactly be measured. In the next section we discuss how

region	function g	function f
$R_{g,f}(0,-)$	including v in g	f is not changed
$R_{g,f}(0,0)$	including v in g	v is included in f
$R_{g,f}(0,1)$	including v in g	v is excluded from f
$R_{g,f}(1,-)$	excluding v from g	f is not changed
$R_{g,f}(1,1)$	excluding v from g	v is included in f
$R_{g,f}(1,0)$	excluding v from g	v is excluded from f

Table 1: Effect of changes in global function of g on global function of f

don't care regions are used to optimize the function of a node while considering the global effects of this change.

3.3.3 Using Don't Care Regions in Node Optimization

During area optimization, the local don't care set of a node is used to optimize the local function of the node for area. During network optimization for power, the don't care information for a node will have to be used to minimize the combination of the power contribution of the node to the network power as well as the switching activity in the transitive fanout nodes.

In a tree network, most nodes have more than one node in their transitive fanouts. This means that while optimizing a node, it is necessary to consider the effect of changes in the function of this node on all nodes in its transitive fanouts. The following lemma gives the number of don't care regions for a node in a tree network.

Lemma 6 *For a node g in a single output tree network with k nodes in its transitive fanout, the number of don't care regions is given by:*

$$2 \cdot \left(3^k - \sum_{j=0}^{k-1} j \cdot \left(3^{k-j} - 1 \right) \right) \tag{11}$$

Proof: The maximum number of don't care regions for node g with respect to its k fanout nodes is given as 2.3^k. This is the possible number of combinations for specifying $R_{g,F}(\alpha, \beta)$ as given in definition 5. For example if $k=2$ then the following enumerates all possible don't care regions for $g=0$:

$R_{g,F}(0,00)$ $R_{g,F}(0,01)$ $R_{g,F}(0,0\text{-})$
$R_{g,F}(0,10)$ $R_{g,F}(0,11)$ $R_{g,F}(0,1\text{-})$
$R_{g,F}(0,\text{-}0)$ $R_{g,F}(0,\text{-}1)$ $R_{g,F}(0,\text{--})$

In a tree network however because of the relationships between observability conditions, some don't care regions will be empty. For example assume $F=(f,h)$ where f is fanout of g and h is fanout of f, Then $R_{g,F}(0,\text{-}0)$ will always be empty. This is because this don't care region refers to points where changes in the value of g does not affect the function of f but it does affect the function of h. This is not possible in a tree network.

Assume the k fanouts of node g specified in F are ordered such that each fanout is listed before all its fanouts. Then don't care regions for $\beta = (\beta_1, \ldots, \beta_{i-1}, \text{"-"}, \beta_{i+1}, \ldots, \beta_k)$ are not empty only if none of the values in $\beta_1, \ldots, \beta_{i-1}$ are "-" and all values in $(\beta_{i+1}, \ldots, \beta_k)$ are all "-". Computing the number of all such conditions and then multiplying it by 2 (for g being 0 or 1) proves the claim of this lemma. ■

By using don't care regions for node g and nodes in its transitive fanout, we can analyze the effect of changes in the function of node g on the signal probability of these fanout nodes.

There are two drawbacks in using the don't care regions to minimize the switching activity of a node and its transitive fanout nodes. The first drawback is that in optimizing node g, we will need information about all don't care regions corresponding to its transitive fanout nodes and since the number of don't care regions grows exponentially with the number of fanouts (lemma 6), this analysis very quickly becomes computationally expensive. A second problem is that contradictory decisions as to increase or decrease the signal probability of a node g can be made while optimizing nodes i_1 and i_2 in its transitive fanin (figure 6). This means that even if all the optimization problems are solved, it is still possible to obtain no improvement because of increasing the signal probability of a node at one step and decreasing it during another step of the procedure.

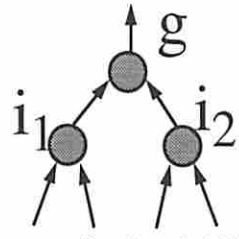

Figure 6: Contradictory decisions on signal probability of g while optimizing i_1 and i_2

The complexity of power optimization procedure is reduced if a decision is made as to only increase (or decrease) the signal probability of a function after it has been optimized. An added advantage of this approach is that conflicting decisions will not be made regarding the new signal probability of this node as nodes in its transitive fanin are being optimized.

The following theorems are used to reduce the complexity of the power optimization procedure.

Theorem 1 *Given a node g and its fanout node f, $R_{g,f}(0,1)$ and $R_{g,f}(1,0)$ are the empty sets and $R_{g,f}(0,0)$ and $R_{g,f}(1,1)$ are maximal if ODC_g is computed as:*

$$ODC_g = ODC_g^f + \overline{f} \cdot ODC_f \tag{12}$$

Proof: The following equations are derived from definition:

$$R_{g,f}(0,0) = \bar{g} \cdot f_g \cdot \overline{f_{\bar{g}}} \cdot ODC_g$$

$$R_{g,f}(0,1) = \bar{g} \cdot f_{\bar{g}} \cdot \overline{f_g} \cdot ODC_g$$

$$R_{g,f}(1,0) = g \cdot f_g \cdot \overline{f_{\bar{g}}} \cdot ODC_g$$

$$R_{g,f}(1,1) = g \cdot f_{\bar{g}} \cdot \overline{f_g} \cdot ODC_g$$

Or equivalently (by substituting $ODC_g = ODC_g^f + ODC_f$):

$$R_{g,f}(0,0) = \bar{g} \cdot f_g \cdot \overline{f_{\bar{g}}} \cdot ODC_f$$

$$R_{g,f}(0,1) = \bar{g} \cdot f_{\bar{g}} \cdot \overline{f_g} \cdot ODC_f$$

$$R_{g,f}(1,0) = g \cdot f_g \cdot \overline{f_{\bar{g}}} \cdot ODC_f$$

$$R_{g,f}(1,1) = g \cdot f_{\bar{g}} \cdot \overline{f_g} \cdot ODC_f$$

which give maximal such conditions. Now if we instead use equation 12 and use ($f=g \cdot f_g + \bar{g} \cdot f_{\bar{g}}$):

$$R_{g,f}(0,0) = \bar{g} \cdot f_g \cdot \overline{f_{\bar{g}}} \cdot ODC_f$$
$$R_{g,f}(0,1) = 0$$
$$R_{g,f}(1,0) = 0$$
$$R_{g,f}(1,1) = g \cdot f_{\bar{g}} \cdot \overline{f_g} \cdot ODC_f$$

which shows that regions $R_{g,f}(0,1)$ *and* $R_{g,f}(1,0)$ *are the empty sets and* $R_{g,f}(0,0)$ *and* $R_{g,f}(1,1)$ *are maximal.* This proves the claim of this lemma.

■

Theorem 2 *Given a node g and its fanout node f,* $R_{g,f}(1,1)$ *and* $R_{g,f}(0,0)$ *are the empty sets and* $R_{g,f}(0,1)$ *and* $R_{g,f}(1,0)$ *are maximal if* ODC_g *is computed as:*

$$ODC_g = ODC_g^f + f \cdot ODC_f \tag{13}$$

Proof: Proof is similar to proof for theorem 1.

■

Theorems 1 and 2 are used as follows. Assume that after optimizing a node *f,* we decide that as other nodes in the network are optimized we want the signal probability of this node to remain the same or only increase beyond its current value. This, for example, is desirable when the signal probability of *f* after it is optimized, is more than 0.5. This condition will disallow any increase in the switching activity of the node beyond its current value. Since a decision is made as to only allow an increase in the signal probability of function *f*, we need to only use regions $R_{g,f}(0,-)$, $R_{g,f}(1,-)$, $R_{g,f}(0,0)$ and $R_{g,f}(1,1)$ while optimizing a node *g* in the fanin of *f*. Using theorem 1 we compute ODC_g such that regions $R_{g,f}(0,1)$ and $R_{g,f}(1,0)$ are empty sets and regions $R_{g,f}(0,0)$ and $R_{g,f}(1,1)$ are maximal. This means that while optimizing node *g*, we know that any use of ODC_g will only result in an increase in the signal probability of node *f* and hence the function of node *g* can be optimized without any concern about adverse effects on switching activity of node *f*. The same approach is used to apply theorem 2 when we want to disallow an increase in the signal probability of node *f* (this is desirable if the signal probability of node is less than *0.5* after it is optimized).

This discussion motivates the definition for *Propagated Power Relevant Observability Don't Care* conditions for a node *f*.

Definition 6 *Given a node f and its fanin node g,* $PPODC_f$ *the Propagated Power Relevant Observability Don't Care for node f is defined as a subset of the observability don't care conditions for f that is used to compute the observability don't care conditions for node g while guaranteeing that any changes in the function of g does not increase the switching activity of node f.* $PPODC_f$ *is defined as follows:*

$$PPODC_f = \begin{cases} \overline{f} \cdot ODC_f & p(f) > 0.5 \\ f \cdot ODC_f & p(f) \leq 0.5 \end{cases} \tag{14}$$

■

3.4 Power Relevant Observability Don't Cares for Low Power Optimization

The goal of don't care computation approach outlined in the previous section is to allow the optimization procedure to concentrate on local changes without concern about global effects on power. This is equivalent to providing maximum flexibility in optimizing a node while guaranteeing that local optimizations do not degrade total power of the network. In this section we use the analysis presented in the previous section to propose a method for computing the set of observability don't care conditions that guarantee that any changes in the function of an internal node using this don't

care set does not increase the switching activity of its immediate fanout node. We then discuss the impact of using this don't care on the switching activity of nodes other than the immediate fanout of the node that is being optimized. In the next section we extend the technique presented here to also guarantee that changing the function of an internal node does not increase the switching activity of any other node in the network.

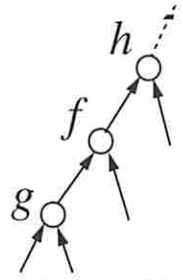

Figure 7: Computing the Power Relevant Observability Don't Care for node g

We provide the following definition for the observability don't care set computed in this section:

Definition 7 *Given a node g and its immediate fanout node f (figure 7), $PODC_g$, the Power Relevant Observability Don't Care for node g is defined as the observability don't care conditions for g that guarantees any changes in the function of g does not increase the switching activity of its immediate fanout node f.*

The following equation is used to computed the power relevant observability don't care for an internal node of Boolean network.

$$PODC_g = ODC_g^f + PPODC_f \tag{15}$$

ODC_g^f gives all the conditions that make changes in g unobservable in the function of f while $PPODC_f$ gives the maximal conditions that guarantee that any changes in the of function g will not increase the switching activity of node f (theorems 1 and 2).

This equation provides a recursive procedure for computing the *PODC* of all nodes in a tree network where each node is optimized after all its fanout nodes. The procedure starts at the primary output of the network where *PODC* is equal to the external don't care defined by the user. After a node f is optimized using $PODC_f$, $PPODC_f$ (using equation 14) and *f_new* (the new function of node f after optimization) are computed. $PPODC_f$ is then stored at node f for future reference. When node g, fanin node of f, is being optimized, $PPODC_f$ that was stored at node f is used to compute $PODC_g$. The same procedure is recursively applied until all the nodes are optimized. Figure 8 shows the procedure for computing the $PPODC_f$ once the function of node f is optimized.

Example 3: Reconsider example 1.

Assume $dc_f = \bar{a}.b$, *also* $p(a)=p(b)=0.5$

Then $PODC_g = PPODC_f + a = f.dc_f + a = 0 + b = a$

After optimization function g is either set to b or is not changed. Regardless of how g is optimized, we see that the switching activity of f does not increase.

■

The procedure presented above guarantees that as a node g is optimized, the switching activity of its immediate fanout node f does not increase (figure7). This procedure does not however make any assumptions on the effect of changing g on the switching activity of nodes in the transitive fanout of

```
1:  function find_propagated_power_odc(new_f, ODC_f)
2:  new_f is the node function after optimization
3:  PODC_f is the Power Relevant observability don't care for f
4:  begin
5:     if ( p(new_f) > 0.5 ) then
6:        PPODC_f = new_f' . PODC_f
7:     else
8:        PPODC_f = new_f . PODC_f
9:     endif
10: end
```

Figure 8: Computing propagated power relevant don't cares

node f (e.g. node h) as explained next.

Lets assume that after node h is optimized, we decide to increase its signal probability when its fanin node is being optimized. This means that don't care regions $R_{f,h}(0,-)$, $R_{f,h}(1,-)$, $R_{f,h}(0,1)$ and $R_{f,h}(1,0)$ are used when optimizing node f. After optimizing node f, the global function of node h changes from 0 to 1 when the function of node f changes from 1 to 0 for set of points $V_{h,1}$ in region $R_{f,h}(1,0)$ and from 0 to 1 for the set of points $V_{h,0}$ in region $R_{f,h}(0,1)$. Also assume after optimizing the function of node f, we decide to increase the signal probability of this node when its fanin node is being optimized. This means that regions $R_{g,f}(0,-)$, $R_{g,f}(1,-)$, $R_{g,f}(0,1)$ and $R_{g,f}(1,0)$ are used in optimizing the function of node g. The function of node f changes from 0 to 1 when the function of node g changes from 1 to 0 for set of points $V_{f,1}$ in region $R_{g,f}(1,0)$ and from 0 to 1 for the set of points $V_{f,0}$ in regions $R_{g,f}(0,1)$. Now if the intersection of $V_{h,1}$ and $(V_{f,0} + V_{f,1})$ is not empty, then changing the function of f from 0 to 1 will result in changing the function of h from 1 to 0 which results in decreasing the signal probability of node h. This analysis shows that it is possible to increase the switching activity of node h while optimizing node g. Note however that any changes in the function of g will not reduce the signal probability of h below its value when it was optimized. This is because the changes in function of g will only affect the function of h in the space defined by don't care regions $R_{f,h}(0,1)$ and $R_{f,h}(1,0)$ and within this subspace, the function of h only evaluates to 1 for points in region specified by $(V_{h,0} + V_{h,1})$ which was only changed from *0* to *1* after node h was optimized.

This analysis shows that after network optimization using the procedure presented in this section, the switching activity of each node in the network is less than or equal to its switching activity immediately after it was optimized. This means that while optimizing different nodes in the transitive fanin of a node f, it is possible to increase or decrease the switching activity of f. However, this new value is always no larger than what it was when node f was first optimized. In the next section we present a method that guarantees a monotone decrease or no change in the current switching activity of all nodes in the network as local node functions are being optimized.

3.4.1. Monotonic Reduction in Global Power

As discussed in the previous section, it is desirable to obtain a monotonic decrease in the global power consumption during node optimizations. The following lemma is used in developing a don't care computation technique that achieves this goal.

Definition 8 *Given a node g and its immediate fanout node f (figure 7), $MPODC_g$, the Monotone Power Relevant Observability Don't Care for node g is defined as the observability don't care conditions for g that guarantee changes in the function of g within this don't care set, do not increase the current switching activity of nodes in the transitive fanout cone of f (excluding node f).*

Lemma 7 *Consider nodes f and h (figure 7). Assume $MPODC_f$ has been computed and f has been optimized using $MPODC_f$ resulting in new_f. $MODC_f$, as defined here, gives the maximal set of conditions that when used in computing the observability don't care conditions for the fanins of f, guarantees that any changes in the transitive fanin cone of f does not increase the switching activity of any node in the transitive fanout cone of f.*

$$MODC_f = ODC_f^h + MPODC_f \cdot \overline{(f \oplus new\_f)} \tag{16}$$

Proof: $(f \oplus new\_f)$ gives all conditions where function of node f has been changed. Note that this change is contained within $MPODC_f$. Therefore, it is also guaranteed that this change has resulted in a decrease in the current switching activity of all transitive fanout nodes where this change is observable. This is a direct consequence of the property defined for $MPODC_f$. For power optimization, if we disallow the function of f to change for all points within $(f \oplus new\_f)$, we are guaranteed that the power reduction obtained in the transitive fanout by changing f will be maintained. This means that $MODC_f$ is obtained by removing the set $(f \oplus new\_f)$ from part of $MPODC_f$ which makes changes in f observable at the transitive fanout cone of f. The part of $MPODC_f$ that makes changes in f observable at the transitive fanout cone of f is:

$$MPODC_f \cdot \overline{ODC_f^h}$$

which is all points in $MPODC_f$ that are not in $ODC^h{}_f$. Then, considering that $ODC^h{}_f \subseteq MPODC_f$:

$$MODC_f = MPODC_f - \{MPODC_f \cdot \overline{ODC_f^h} \cdot (f \oplus new\_f)\}$$

$$= MPODC_f \cdot \{\overline{MPODC_f} + ODC_f^h + \overline{(f \oplus new\_f)}\}$$

$$= ODC_f^h + MPODC_f \cdot \overline{(f \oplus new\_f)}$$

This proves the claim of this lemma.

■

This lemma provides us with a method for computing the observability don't care conditions for node g (Figure 7) that guarantees that changes in g will not increase the switching activity of transitive fanout nodes of f. Using the result of this lemma, the following theorem provides a don't care computation technique that guarantees a monotonic reduction in the power consumption of all nodes in the network.

Theorem 3 *Consider nodes g, f and h (figure 7). Assume f has been optimized using $MPODC_f$ resulting in new_f where new_f is the global function of node f after optimization. $PMPODC_g$, Propagated Monotone Power Relevant Observability Don't Care for node f as defined here, gives the maximal set of don't care conditions that when used in optimizing node g, guarantees that any changes in the function of g does not increase the current switching activity of node f or any node in its transitive fanout cone.*

$$PMPODC_f = \begin{cases} \overline{new\_f} \cdot \left(ODC_f^h + \overline{f} \cdot MPODC_f \right) & p(f) > 0.5 \\ new\_f \cdot \left(ODC_f^h + f \cdot MPODC_f \right) & p(f) \leq 0.5 \end{cases} \tag{17}$$

Proof: We prove the first part for when $p(f) > 0.5$. The proof for the second part is similar.

Since the signal probability of f is greater than 0.5, we need to propagatepart of $MPODC_g$ such that the signal probability of f does not decrease for any changes in function of g. We also need to use part of $MPODC_f$ that guarantees that changes in g do not result in an increase in the power consumption of nodes in the transitive fanout of f. Using theorem 1 and lemma 7 we can write:

$$PMPODC_f = \overline{new\_f} \cdot MODC_f$$

Using equation 16, expanding the exclusive or term and simplifying the expression proves the claim of this theorem.

■

Given a node g and its fanout f, $MPODC_g$ the monotone power relevant observability don't care for node g (which is used to optimize the function of g) is calculated using the following equation:

$$MPODC_g = ODC_g^f + PMPODC_f \tag{18}$$

Once $MPODC_g$ is used to find *new_g* (the new function for node *g*), $PMPODC_g$ (propagated monotone power relevant observability don't care for *g*) is calculated using the procedure shown in figure 9 and stored at node g for computing the don't care conditions for fanins of g.

```
1:  function find_propagated_monotone_power_odc(f, new_f, ODC^h_f, MPODC_f)
2:  f is the node function before optimization
3:  new_f is the node function after it is optimized
4:  ODC^h_f is the observability don't care for f at the output h
5:  MPODC_f is the monotone power relevant ODC for f
6:  begin
7:      if (p(new_f) > 0.5 ) then
8:          PMPODC_f = new_f'. (ODC^h_f + f' . MPODC_f)
9:      else
10:         PMPODC_f = new_f. (ODC^h_f + f. MPODC_f)
11:     end if
12: end
```

Figure 9: Don't care filter for monotonic decrease in current switching activity

The advantage of using the techniques proposed in this section is that it guarantees that local optimizations not only improve the local power, but also work on reducing the switching activity of other nodes in the network when possible. These filters however result in a smaller don't care sets while optimizing the local function of a node and this means that it may not be possible to obtain as much reduction in the power contribution of the node to the network power if the power don't care filters were not being used. Therefore care should be taken in using these filters in order to keep a balance between the available don't care for local optimizations and the don't care being used to guarantee an overall reduction in the global power due to any local changes.

3.5 Analysis for DAGs

The approach used to analyze a tree network is directly used to analyze a general Boolean network. However since the observability don't care relations for trees do not hold in a general DAG, the number of don't care regions for a node is larger than the number of don't cares regions for trees. The upper bound on the number of don't care regions for a node g in a DAG is given by $2 \cdot 3^k$ where k is the number of transitive fanout nodes of g. This for example is possible if node g has k immediate fanout nodes.

The don't care used for optimizing node g in a DAG is calculated using the following equation.

$$PODC_g = \prod_{f_i \in \text{ fanouts(g)}} PODC_g\left(f_i\right) \tag{19}$$

where $PODC_g(f_i)$ gives the power relevant observability conditions for node g through fanout edge f_i and is computed using equation 15. $MPODC_g(f_i)$ computed using equation 18 can also be used to compute $MPODC_g$. However, $PMPODC_{fi}$ for each fanout f_i of g is computed using the following equation derived by extending lemma 7 and theorem 3.

$$PMPODC_{fi} = \begin{cases} \overline{new\_f_i} \cdot \left(\left\langle \prod_{h_{ij} \in \text{ fanouts(fi)}} ODC_{fi}^{hj} \right\rangle + \overline{f_i} \cdot MPODC_{fi} \right) & p(f) > 0.5 \\ new\_f_i \cdot \left(\left\langle \prod_{h_{ij} \in \text{ fanouts(fi)}} ODC_{fi}^{hj} \right\rangle + f_i \cdot MPODC_{fi} \right) & p(f) \leq 0.5 \end{cases} \tag{20}$$

Note that equations 15 or 18 provide maximal power relevant don't care sets for tree networks. This maximality does not however hold when equation 19 is used [21].

The analysis presented in this section computes the maximal set of power relevant observability don't cares for nodes in a tree network. In general it is desired to compute power relevant observability don't care sets that can be used to optimize all nodes without recomputing the don't care sets.

The equations for computing the power relevant observability don't cares have been derived by modifying the part of don't care that is propagated in the network (the second term in equation 6) without making any assumption about the first part in this equation which gives observability conditions for immediate fanout nodes. The operations to generate compatible don't care sets however modify the first term in equation 6 without making any assumptions on the don't care being propagated. The equations for computing power relevant don't cares are easily extended to generate compatible (in the sense that they can be used to optimize nodes without recomputing the don't care sets) by replacing ODC^f_g in equations 15 and 18 with that computed in equation 7.

Using the procedures presented in this section, we calculate the set of *maximal compatible power relevant local don't care* or *maximal compatible monotone power relevant local don't care* for nodes in the network which is then used to optimize the local function of nodes without any concern on degrading the global power consumption of the network. In the next section we present a new approach for including the *SDC* in the don't care of the function being optimized.

3.6 Power Relevant Satisfiability Don't Cares for Low Power Optimization

In a Boolean network, some combinations for the values of the internal nodes are not possible no matter what input vector is applied at the primary inputs of the circuit. If a network has n primary input nodes and m internal nodes then satisfiability don't care conditions (*SDC*) for a network contain all impossible combinations in the space of B^{n+m}. The contribution of each node in the network to *SDC* of network is given in definition 2. In this sense we define *SDC* due to a node g as $g \oplus F(g)$ where g is the variable at the output of node g and $F(g)$ is the function of node g in terms of its immediate fanins. Satisfiability don't cares are usually used to substitute a new variable into a function if this substitution results in a lower cost implementation. The following example illustrates this idea.

Example 4:

Consider the Boolean network with functions f and h where:

$$f = a.b + \overline{a}.\overline{b}$$
$$g = \overline{a}.b + a.\overline{b}$$

While optimizing node f, SDC due to node g can be included in the don't care set of f. SDC due to node g is given as $g \oplus (\overline{a}.b + a.\overline{b})$. *Optimizing function f using this SDC will result in the following form for f:*

$f = \bar{g}$

where g is successfully substituted in the function of f.

■

While optimizing a function f in the network, we usually use a subset of *SDC* for nodes that with high probability may be substituted into f. In [21] a method for selecting a subset of *SDC* is presented where only *SDC* of nodes whose immediate support is a subset of the immediate support of the node being optimized is considered.

It has been shown [17] that using *SDC* due to any node in the network for simplifying the function of a node f does not result in changing the global function of f or any other node in the network. This means that using *SDC* does not change the signal probability or switching activity of any of the nodes in the network. Therefore *SDC* may freely be used to optimize the function of nodes without concern that switching activities may increase. Successful use of *SDC* may however result in using a new variable v in the function of the node being optimized. This results in a load increase at the output of node n_v generating variable v. If the switching activity of the new variable is high, then increasing load on this variable may result in an unexpected increase in the power consumption of the network. It is therefore important to take into account the switching activity of nodes that are being considered for possible substitution in the function of node that is being optimized.

The approach presented in [21] for using *SDC* consists of identifying a set of nodes that with high probability may be substituted in node f being optimized. These nodes are identified as all nodes whose immediate support is a subset of the immediate support of node f. The *SDC* due to each of these candidate nodes is then included in the don't care of f (see example 4). For power optimization, we use the following approach for finding the set of candidate nodes whose *SDC* will be included in the don't care of a node f. Figure 9 shows the transitive fanin and fanout nodes of node f in a Boolean network. *SDC* of nodes in the transitive fanout of node f cannot be included in the don't care of f. All other nodes in the network may be substituted into f if their primary input support is a subset of the primary input support for node f. Nodes g, m, n and p show such candidate nodes. In order to select a set of candidate nodes, we first find all nodes that are not in the transitive fanout of f and whose primary input support is a subset of the primary input support of f. Among these nodes we then select nodes whose switching activity is below a user defined threshold value.

The SDC due to a node g (see figure 9) whose immediate support is a subset of the immediate support for node f is easily included as $g \oplus F(g)$. In order to include the *SDC* due to nodes that do not share the same immediate support as f (node m in figure 9), it is necessary to include the *SDC* for all nodes that are in the transitive fanin cone of f and transitive fanin cone of m. This is necessary since successful substitution of m into f requires that the unsatisfiable conditions relating the values at the output of node m and all immediate fanins of f be known. The first problem with this approach is that including the *SDC* for all these node will result in also considering them for possible substitution into f while f is being optimized and this may not be desirable if these nodes have a high switching activity. The second drawback is that this operation may prove to be expensive when the number of nodes in the transitive fanin cones of f and m are large. An alternative approach for including *SDC* of nodes that do not share the same immediate support as f is proposed here by observing that given a Boolean network with n primary inputs and m internal nodes, the range of B^n, space of primary inputs onto B^{n+m}, the space of all nodes in the network, gives the set of all satisfiable conditions in the network.

Assume an internal node f with fanins $\{i_1, ..., i_l\}$ *is* being optimized while considering possible substitution of nodes $\{n_1, ..., n_k\}$ into the function of f. The complement of the range of the space of primary inputs by function $F=\{i_1, ..., i_l, n_1, ..., n_k\}$ gives all unsatisfiable conditions in the space of function F which is used as the *SDC* set while optimizing node f. Using this technique, it is no longer necessary to include the *SDC* due to nodes which are not good candidates for substitution into function of node f.

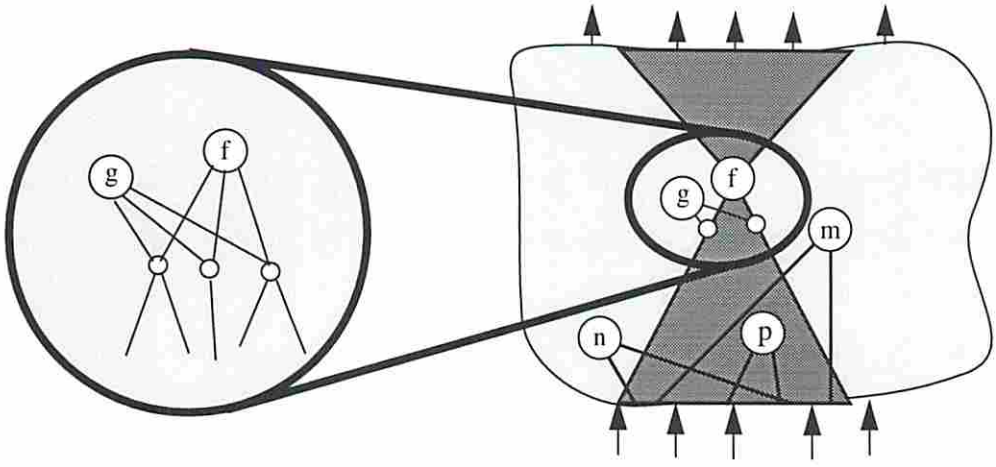

Figure 10: Candidate nodes for SDC computation

The procedures presented in this section compute a set of local don't care for the function that is being optimized. This local don't care guarantees that global power is not degraded while the node is being optimized and also allows for expressing the node function using a new variable which may potentially result in a lower power consumption. In the next section technique using minimal literal and variable supports are presented to optimize the local function of a node for low power.

4 Node Optimization using Minimal Variable Supports

The goal of node optimization for power is to minimize the power contribution of the node to the overall power consumption of the network. This requires that the combination of the node power at the input and output as well the estimate for the internal power of the node be minimized. In this section we present a method for minimizing the power of the node by reducing load on high activity inputs of the network. In this section we first present a more efficient method for computing the set of minimal literal and variable supports of the nodes in the network. The set of minimal supports gives the flexibility in implementing the node function using different sets of variables. We then propose techniques for selecting a node support which will potentially lead to maximal reduction in the node input power.

4.1 Minimal Supports of Functions

4.1.1 Minimal Literal Supports

Given an incompletely specified function *ff*, it is often possible to implement *ff* using different sets of literals. For example let $F = a.b$ and $D_F = a \oplus b$. This function can be simplified to $F = a$ or $F = b$. Then set $\{\{a\}, \{b\}\}$ is the set of all minimal literal supports for node F.

The problem of finding the minimal literal support of a function is stated as follows.

Problem: *Given* $C^1=\{C^1_0, C^1_1,\ldots, C^1_g\}$, *the cover of the on-set and* $C^0=\{C^0_0, C^0_1,\ldots, C^0_h\}$, *the cover of the off-set of a function* $F(x_1, x_2,\ldots x_n) \in R^n$, *find the set all minimal literal supports of the function F.*

The following procedure for finding the set of all minimal dependence sets is presented in [9]. Each of these sets gives the set of literals necessary for implementing the function.

Algorithm 1:

- The problem is first transformed into the R^{2n} space. This is done by replacing the complement of variable x_i in the on-set with a new variable x_{i+n}. The positive occurrences of variable x_i in the cubes of the off-set is replaced with the complement of the new variable

x_{i+n}. A new function $f(x_1, x_2, ...x_{2n}) \in R^{2n}$ is obtained with cover $c^1=\{c^1_0, c^1_1, ..., c^1_g\}$ for the off-set and cover $c^0=\{c^0_0, c^0_1, ..., c^0_h\}$.

- For each cube c^1_i of the on-set and c^0_j of the off-set of function f define:

$$H_{ij} = \sum \langle x_k | x_k \supseteq c_i^1, \bar{x}_i \supseteq c_j^0 \rangle. \tag{21}$$

This function corresponds to the union of the set of literals where the presence of each of literal will result in an empty intersection between cubes c^1_i and c^0_j.

- Define function H as:

$$H = \prod_{\substack{i \in 1 \ldots g \\ j \in 1 \ldots h}} H_{ij}.$$

The cubes of function H give the minimal support sets of function f in R^{2n}.

- The set of variables in R^{2n} is transformed back to R^n by replacing variables n+1 through 2n with the complement of the variables 1 through n. Performing this operation on functions f and H will result in functions F and the minimal literal support for function F.

■

This procedure is explained as follows. H_{ij} for cubes c^1_i and c^0_j gives the set of literals, one of which need to remain lowered in c^1_i in order for this cube not to intersect cube c^0_j of the off-set. This is represented as a conjunctive term. The intersection of all H_{ij} gives all the conditions such that none of the cubes of on-set intersect any of the cubes in the off-set. The reason the problem is transformed into R^{2n} space is that each variable in the R^{2n} space represents the logical event that a positive or negative literal of the function inputs remain lowered. This means that if a minimal literal support requires both positive and negative phases of a variable, then the cube representing this support will not be removed by using the relation $x.x = 0$. Note that H is a unate function in all the variables in the R^{2n} space and therefore has a unique minimum form representation which consists of all prime implicants of H. Also note that if function F is positive (negative) unate in a variable v, then it is not necessary to introduce a new variable for the negative (positive) literal of variable v. Therefore if a Function F has n inputs and is unate in m variables, then it is sufficient to transform the problem into R^{2n-m} space. The prime implicant cubes of H correspond to the set of minimal literals that need to remain lowered for the on-set not to intersect the off-set.

Once the set of all minimal literal supports of a function has been computed, a literal support is selected to implement the function. This problem is stated as follows:

Problem: *Given $C^1=\{C^1_0, C^1_1, ..., C^1_g\}$, the cover of the on-set for a function $F(x_1, x_2, ... x_n) \in R^n$, and a set of minimal literal support given as a set of literals $MLS_F=\{lit_1, lit_2, ..., lit_k\}$, find the minimal irredundant form of the function F.*

The solution to this problem is obtained by raising all literals in on-set of F which are not a member of set MLS_F [9]. The following example shows how these algorithms are used.

Example 5:

Consider the following incompletely specified function F where g =2 and h =1:

$$on\text{-}set(F) = \bar{x}_1\bar{x}_2x_3x_4 + \bar{x}_1x_2x_3\bar{x}_4$$
$$off\text{-}set(F) = \bar{x}_1x_2\bar{x}_3x_4$$

by setting: $x_5=\bar{x}_1$, $x_6=\bar{x}_2$, $x_7=\bar{x}_3$, $x_8=\bar{x}_4$ we will have:

$$on\text{-}set(f) = x_5x_6x_3x_4 + x_5x_2x_3x_8$$
$$off\text{-}set(f) = \bar{x}_1\bar{x}_6\bar{x}_3\bar{x}_8$$

Therefore:

$H_{11} = x_6 + x_3$ $\quad H_{21} = x_8 + x_3$ $\quad H = x_6.x_8 + x_3$

The set of all minimal literal supports of F is given as:

$\{\bar{x}_2.\bar{x}_4\}, \{x_3\}$

This means that two possible implementation of F are:

$F = x_3$ $\quad$ *or* $\quad F = \bar{x}_2 + \bar{x}_4$

■

4.1.2 Using Reduced Off-sets in Computing Minimal Literal Supports

The method described in [9] for computing the set of all minimal literal supports requires that a cover of the on-set and off-set of the function be computed. The off-set has to be computed by complementing the union of on-set and don't care of the function. This operation is in general computation expensive and the resulting off-set might have an exponential size. An example of this function is the Achilles Heel function which has n terms in the cover of on-set and 3^n terms in the cover of off-set. Therefore it is desirable to compute the set of all minimal literal supports without computing the off-set of the function. In this section we present a method for computing the set of all minimal literal supports of a function without computing the off-set by using the ideas behind reduced off-sets.

Reduced off-sets are introduced by observing that some minterms of the on-set or don't care cannot be used to expand a cube of the on-set. Assume $p=\bar{a}.\bar{b}$ and the complete off-set is $a \oplus b$. Then the reduced off-set of p is $(a + b)$ which is all that is needed to expand p.

Definition 9 *(Malik [16])Given a cube p of a function f, R_p, the reduced off-set of function f with respect to cube p is obtained by including all minterms of the on-set that cannot be used to expand p, in the off-set of the function.*

It is also shown in *[16]* that the reduced off-set of a cube is a unate function and therefore has a unique minimal representation in the SOP form. A procedure is presented in *[16]* for computing the reduced off-set of each cube in the function where reduced off-sets are computed without computing the complete off-set of the function.

The following theorem shows how reduced off-sets are used to compute the set of all minimal literal supports of a function.

Theorem 4 *Given a cube $c^1{}_r$ of the cover of the on-set of an incompletely specified function and R_r corresponding reduced off-sets for $c^1{}_r$, H_r, the set of all minimal literals supports of $c^1{}_r$ is given by:*

$$H_r = \overline{R_r} \tag{22}$$

Proof: Given $\{c^0{}_1,..c^0{}_h\}$, a cover of the off-set for f, define:

$$H_r = \prod_{j=1}^{h} H_{rj} \tag{23}$$

By definition:

$$H_r = \prod_{j=1}^{h} \sum_{k} \langle x_k | x_k \supseteq c^1_r, \bar{x}_k \supseteq c^0_j \rangle$$

$$\overline{H_r} = \sum_{j=1}^{h} \prod_{k} \langle \bar{x}_k | x_k \supseteq c^1_r, \bar{x}_k \supseteq c^0_j \rangle = R_r$$

The last equation is by definition equal to the reduced-off-set for cube $c^1{}_r$. Note that the reduced off-set for a cube p of the on-set if unate in all variables. Therefore it is not necessary to transform

the problem into R^{2n} space when finding the support for a single cube of the on-set.

■

Theorem 5 *Given $F = \{c^1_1, c^1_2, ..., c^1_g\}$ a cover of the on-set and $\{R_1, R_2, ...R_g\}$ the set of corresponding reduced off-sets for an incompletely specified logic function $f(x_1,...x_n)$, the set of all minimal literals supports of f is given by:*

$$H = \prod_{r=1}^{g} \overline{RR_r}$$

where RR_r is obtained by transforming R_r into R^{2n} space as described in algorithm 1.

Proof: By definition:

$$H = \prod_{r=1}^{g} H_r.$$

where H_r is as defined in equation 23. Theorem 4 also showed that $H_r = \overline{R_r}$. This means that H can be computed by taking the intersection of the minimal literal supports of each cube of the on-set. The problem however is that even though each R_r is unate in all variables, the set of all reduced off-sets are not unate with respect to the same sense of the variables. For example R_i may be positive unate with respect to variable v and R_j may be negative unate with respect to the same variable. This means that it is necessary to transform each reduced off-set to the R^{2n} space before the intersection of all minimal literal supports has been computed.

■

```
1:   function Generate_MLS(F)
2:   F is a Boolean function with cubes (c1,...cg)
3:   begin
4:       SupBar = 0
5:           foreach ( cube cr of the cover) do
6:           Rr = findReducedOffset(F, cr)
7:           RRr = transformToNewSpace(Rr)
8:           SupBar = SupBar + RRr
9:       endfor
10:      Support = ¬SupBar
11:      return Support
12:  end
```

Figure 11: Minimal Support using Reduced off-sets

This approach greatly reduces the complexity of computing the set of all minimal literals supports of an incompletely specified function when the size of don't care set is large. Using this theorem, the procedure shown in figure 11 presents an algorithm for generating the set of all minimal literal supports of an incompletely specified function.

4.1.3 Minimal Variable Supports

The procedure in section 4.1.2 computes the set of all minimal literal supports of a function. For some optimization procedures, it may not be necessary to differentiate between the positive and negative literals of a variable. This means that it is sufficient to compute the set of all minimal variable

supports of the function.

The advantage of computing the variable support is that it is no longer necessary to transform the problem into R^{2n} space where n is the number of variables. The following theorem provides a method for computing the set of all minimal variable supports.

Theorem 6 *Given $F = \{c^1_1, c^1_2, ..., c^1_g\}$ a cover of the on-set and $\{R_1, R_2, ...R_g\}$ the set of corresponding reduced off-sets for an incompletely specified logic function $f(x_1,...x_n)$, the set of all minimal literals supports of f is given by:*

$$H = \prod_{r=1}^{g} \overline{\hat{R}_r} \tag{24}$$

where $\hat{R}_r$ is obtained by replacing all positive literals in R_r with negative literals.

Proof: The reason that in theorem 5, reduced off-sets are transformed into R^{2n} space for literal support computation is that if a literal support has both the positive and negative literals of a function, then the Boolean operations defined for finding the set of minimal literal supports will effectively remove any such support from the solution since a support is represented as a cube containing both positive and negative occurrence of the literal. During variable support computation, each reduced off-set R_r gives the set of minimal literal supports of cube c^1_r. By making all literals negative, then all reduced off-sets are negative unate in all variables which means that it is no longer necessary to transform the problem into R^{2n} space. The literals are also changed into negative literals so that after complementing the reduced off-sets, the minimal variable supports of the function are represented as positive literals.

■

4.2 Node Minimization using Minimal Variable Supports

Once the set of minimal variable supports for a function is computed, a decision has to be made as to which set of variables to use for implementing the function. A simple cost function for minimizing power consumption is to count the number of variables in the variable support. The drawback with this cost function is that it does not consider the switching activity of fanin variables that constitute the support variables. A better cost function is to choose a variable support where the sum of the switching activity for all the variables in the support is minimum. We refer to this procedure as the "*minimal switching activity support*" procedure. Once the new variable support for a node is determined, the new function of the node is computed by dropping variables not in the support.

When a variable support is selected for the function, a part of the don't care is assigned to eliminate the variables not in the selected support of the node. This operation results in a new function f_{new}. However a subset of the don't care can still be used to minimize the cover of f_{new}. This subset of the don't care is called reduced don't care $dc_{reduced}$.

Theorem 7 *Given a cube v representing the variables removed from the on-set of a function f and dc, don't care for function f, $dc_{reduced}$ the reduced don't care for f, as given below, is the maximal set of don't care that can be used to optimize f without including variables from v in f.*

$$dc_{reduced} = C_{\bar{v}}(dc)$$

where $\bar{v}$ is the bit-wise complement of *v*.

Proof: Given a function F and a set of variables v, then $C_v(F)$ gives the largest function contained in F which is independent of variables in the set v. The proof for this theorem follows from the observation that $C_v(dc)$ gives the maximal set of don't cares contained in *dc* which is independent of variables in v.

■

Figure 12.a shows the on-set and don't care for a function *f*. Figure 12.b shows the don't care

assignment which is used to eliminate variable x from the support to obtain f_{new} and figure 12.c shows the reduced don't care for function f after variable x is eliminated from the k-map. Using reduced don't care for this node, one product term is removed from the on-set of the function f. Given a function f, its don't care set dc and a variable support v, the procedure in figure 13 is used to optimize the power consumption of the function.

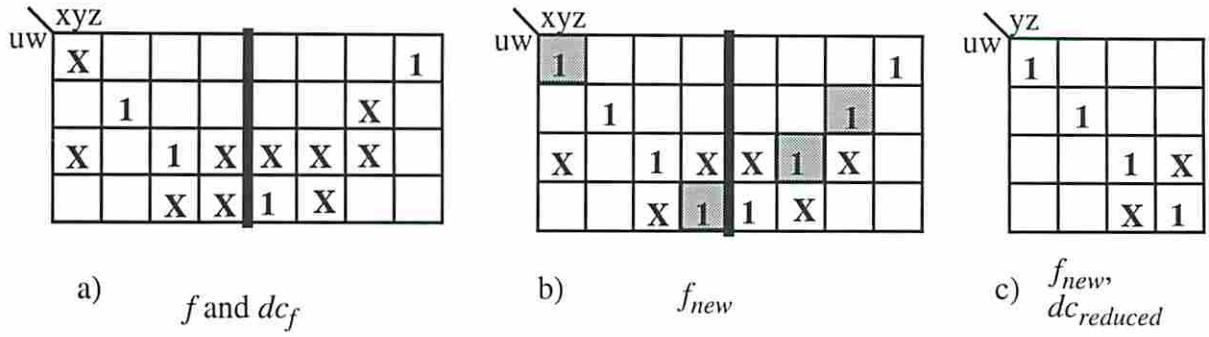

Figure 12: Using the reduced don't care

```
1:  function node_fanin_optimize(f, dc, v)
2:  f is the function and dc is the don't care of node, v is a variable support;
3:  begin
4:      p= cube representing eliminated variables
5:      f_reduced = function_elim_variables(f, p)
6:      dc_reduced = C_p(dc)
7:      f_new = espresso(f_reduced, dc_reduced)
8:      return f_new
9:  end
```

Figure 13: Node minimization using variable support v

The given procedure will provide a low area implementation which has the lowest sum of switching activities on the immediate fanins of the node. It is however possible for a variable support with a higher switching activity support cost to have a smaller factored form and hence have a lower power. In order to select a variable support which also reduces the node's power estimate as much as possible, we compare the power estimate for the node implementation of the k lowest cost variable supports where k is a user defined parameter. Note that our node power cost function in the factored form as given in equation 3, also takes into account the output load and switching activity. Hence by selecting the lowest power cost implementation, we will select a solution which minimizes all factors contributing to the network power consumption. Given a node function f and its don't care dc, the procedure in figure 14 is used to select the lowest power cost implementation of the function.

5 Results

The procedures presented in this paper were implemented in a program called *power_full_simplify* and the results for example benchmarks were compared to those of the *full_simplify* command in the *SIS* package.

Each example in the benchmark set is first optimized using script.ruuged and then mapped for minimum power. The same example is also optimized using the power script (where full_simplify is

```
1:  function node_power_optimize(f, dc)
2:  f is the function and dc is the don't care of node;
3:   begin
4:      f_esp = espresso(f, dc).
5:      V = find_k_minimal_switching_activity_var_sup(f, dc).
6:      foreach v ∈ V do
7:          f_tmp = node_fanin_optimize(f, dc, v)
8:          if( power_cost(f_tmp) < power_cost(f_esp) ) then
9:              f_esp = f_tmp
10:         endif
11:     endfor
12:     return f_new
13: end
```

Figure 14: Node optimization for power

replaced with power_full_simplify) and then mapped for minimum power.

Table 2 shows the results of the optimization after running the *script.rugged* on benchmark examples and then mapping for minimum area. Columns 2 and 3 give the area and power of the network right after the full_simplify command. The area is given as the number of literals in the factored form and the power is estimated using the factored form model. Columns 4, 5 and 6 give the network area, delay and power after technology mapping using library *lib2.genlib* where area, delay and power are computed using the library parameters. Power is also measured under a zero delay model and randomly set input signal probabilities.

The results in table 3 are generated by replacing the *full_simplify* command in the *script.rugged* with the *power_full_simplify* command and then mapping for minimum power. All results are normalized with respect to results in table 2. Again, columns 2 and 3 give the area and power estimates before mapping and column 4, 5 and 6 give the area, delay and power of the mapped networks computed using the library parameters. As results show, on average we were able to obtain %10 improvement in power with %4 reduction in circuit area. On average the circuit delay has been increased by %5.

Tables 4 shows the runtimes for full_simplify and power_full_simplify commands. As the results show, power_full_simplify is on average 2 times slower than full_simplify. In table 5, column 2 shows the number of nodes in each example, column 3 shows the average number of inputs considered for each node in a given example and column 4 gives the average number of minimal variable supports found for each node in the example.

We expect to obtain better results if external don't cares are given for the circuits under consideration. The circuits we used had no external don't cares; consequently, the *ODC* for these networks were usually small.

6 Conclusion

In this paper a method is presented that allows for minimizing the power consumption of a network using the don't care conditions and local function minimization. Using the techniques presented here it is possible to guarantee that local node optimization will not increase the power consumption in the transitive fanout nodes. This means that local nodes can be optimized without concern for how changes in the function of the current node affect the power consumption in the rest of the network. A method for optimizing the local function of node was also presented where the concept of minimal lit-

eral and variable supports have been used to find the lowest cost input supports and then lowest cost implementation of the function.

Example	Before Mapping		After Mapping		
	Area(factored	Power	Area	Delay	Power
5xp1	113	41.54	118784	31.96	3.60
Z5xp1	116	47.39	122032	34.62	4.19
9sym	211	96.23	226896	21.76	8.21
9symml	186	73.64	205552	22.51	6.10
apex5	777	114.94	934032	41.38	9.87
apex6	743	268.51	814320	25.13	24.76
apex7	245	80.53	266800	20.44	7.44
b12	79	19.28	88160	12.79	1.80
bw	158	38.93	170288	40.79	3.86
clip	132	59.46	147552	21.90	4.83
cps	1237	219.53	1379008	40.67	19.15
des	3462	1077.95	3691584	173.06	95.61
duke2	446	96.13	498800	33.42	8.35
e64	253	34.05	294176	111.02	2.62
ex5	345	89.20	345216	26.86	8.11
example2	331	80.15	362384	19.31	6.98
frg2	886	187.66	875568	42.01	16.34
k2	1135	114.05	1243056	33.13	9.85
misex1	52	15.50	56608	14.28	1.47
misex2	103	22.09	114144	12.43	2.14
pair	1600	504.70	1676432	43.96	43.42
pdc	410	119.24	437552	21.33	10.36
rd84	145	48.60	161472	19.77	4.12
rot	671	204.30	719664	31.54	19.60
spla	648	160.79	756320	25.37	13.30
squar5	56	12.22	59392	23.68	1.08
t481	881	79.68	813856	27.90	6.68
ttt2	219	61.13	232464	17.73	5.29

Table 2: Area, delay and power statistics for area script.

Examples	Before Mapping		After Mapping		
	Area(factored)	Power	Area	Delay	Power
5xp1	0.93	0.98	0.86	0.82	0.90
Z5xp1	0.97	0.91	0.95	0.78	0.84
9sym	0.89	1.01	0.83	0.86	0.87
9symml	1.24	1.02	1.15	1.12	0.84
apex5	0.99	0.96	0.96	0.93	0.95
apex6	0.99	0.96	1.00	1.24	0.96
apex7	0.99	0.99	0.97	1.22	0.98
b12	0.96	0.99	0.99	0.92	0.96
bw	1.01	0.87	1.01	0.80	0.90
clip	1.02	0.91	0.94	1.16	0.91
cps	0.99	0.97	1.00	1.17	0.97
des	1.01	1.01	1.02	1.07	0.99
duke2	1.01	1.01	0.99	1.13	0.97
e64	1.00	0.51	0.83	1.16	0.50
ex5	0.99	0.89	0.99	0.97	0.91
example2	0.99	0.96	1.01	1.03	0.97
frg2	0.94	0.86	0.94	1.02	0.86
k2	0.99	0.91	0.98	1.14	0.90
misex1	1.00	0.90	0.96	1.22	0.94
misex2	1.00	0.87	0.97	1.50	0.91
pair	0.99	0.98	0.99	1.04	0.97
pdc	1.00	0.90	1.00	1.09	0.89
rd84	0.97	0.85	0.84	1.13	0.76
rot	0.98	0.94	0.96	1.10	0.95
spla	1.00	0.93	0.99	1.10	0.91
squar5	0.93	0.89	0.86	0.70	0.87
t481	1.00	0.97	1.00	1.01	0.98
ttt2	0.97	0.91	0.90	1.11	0.88
Average	0.99	0.92	0.96	1.05	0.90

Table 3: Area, delay and power statistics for power script (normalized with respect to results for the the area script).

Example	area script (seconds)	power script (seconds)	Ratio power/area
5xp1	2.2	7.5	3.41
Z5xp1	2.1	8.4	4.00
9sym	22.6	21.4	0.95
9symml	31.1	36.5	1.17
apex5	25.8	40.3	1.56
apex6	11.3	27.4	2.42
apex7	3.4	6.3	1.85
b12	0.9	2.0	2.22
b9	1.0	3.0	3.00
bw	5.5	7.7	1.40
clip	9.3	29.8	3.20
cps	331.2	375.6	1.13
des	355.1	496.2	1.40
duke2	47.6	87.6	1.84
e64	20.7	35.0	1.69
ex5	13.2	27.9	2.11
example2	4.4	8.5	1.93
frg2	61.6	81.4	1.32
k2	114.0	127.4	1.12
misex1	0.5	1.3	2.60
misex2	0.9	2.1	2.33
pair	69.3	159.2	2.30
pdc	314.1	319.0	1.02
rd84	15.1	21.2	1.40
rot	26.9	69.2	2.57
spla	234.3	318.6	1.36
squar5	0.7	2.4	3.43
t481	215.8	226.1	1.05
ttt2	2.9	8.1	2.79
vda	27.4	38.1	1.39
Average			2.00

Table 4: Runtime for the area and power scripts.

Example	Number of nodes	average variables/node	average variable supps/node
5xp1	16	6.38	3.06
Z5xp1	16	5.88	3.94
9sym	18	3.72	1.72
apex5	168	6.96	1.68
b12	15	5.27	1.53
bw	34	7.26	2.35
clip	23	6.09	4.83
cps	261	5.95	1.71
duke2	80	6.30	2.10
e64	125	5.39	1.52
ex5	92	6.25	2.86
misex1	10	6.10	1.90
misex2	26	5.81	1.81
pdc	116	6.47	2.19
rd84	23	4.48	3.22
spla	140	6.10	2.25
squar5	9	6.33	3.56
9symml	18	5.50	4.17
apex6	147	6.01	2.32
apex7	65	4.94	1.55
b9	32	4.91	1.94
des	584	6.86	3.32
example2	91	4.75	1.35
frg2	223	6.13	1.94
k2	307	5.88	1.50
pair	294	6.59	1.53
rot	168	5.01	1.36
t481	290	4.78	1.41
ttt2	48	5.50	2.29
vda	165	5.89	1.48
Avg		5.78	2.28

Table 5: Average node variable Supports for each example.

Reference

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh and M.Papaefthymiou. "Precomputation-based Sequential Logic Optimization for Low Power." In IEEE Transactions on VLSI Design, volume 2, pages 426-436, December 1994.

[2] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli. "Multi-Level Logic Synthesis." In proceedings of the IEEE, 78(2):264-300, February 1990.

[3] R. Brayton, G.D. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli. "Logic Minimization Algorithms for VLSI Synthesis," Kluwer Academic Publishers, Boston, 1984.

[4] J. B. Burr. Stanford ultra low power CMOS. In Proceedings of Hot Chips Symposium V, pages 583–588, June 1993.

[5] E. Cerny. An approach to unified methodology of combinational switching circuits. IEEE International Conference on CAD, 27:8, August 1977.

[6] A. P. Chandrakasan, S. S. Scheng, and R. W. Broderson. Low power CMOS digital design. IEEE Journal of Solid State Circuits, 27(4):473–483, April 1992.

[7] O. Coudert, C. Berthet, and J.C. Madre. :Verification of Sequential Machines based on Symbolic Execution." In proceedings of the Workshop on Automatic Verification Methods for Finite State Systems, Grenoble, France, 1989.

[8] M. Damiani, G. De Micheli. "Observability Don't Care Sets and Boolean Relations." In IEEE International Conference on Computer Aided Design, pages 502-505, November 1990.

[9] C. Halatsis and N. Gaitanis. Irredundant normal forms and minimal dependence sets of a boolean function. IEEE Transaction on Computers, pages 1064–1068, November 1978.

[10] J. Hong, R. G. Cain, and D. L. Ostapko. "MINI: A heuristic approach for logic minimization." In *IBM journal of Research and Development*, volume 18, pages 443–458, September 1974.

[11] S. Iman. "Technology Independent Logic Synthesis for Low Power." Ph.D. thesis proposal manuscript. University of Southern California. May 1995.

[12] S. Iman and M. Pedram. ``Multi-level network optimization for low power,'' In Proc. of the IEEE Int'l. Conf. on Computer Aided Design, pages~372-377, November 1994.

[13] S. Iman and M. Pedram. "Logic extraction and factorization for low power." In proceedings of the Design Automation Conference, June 1995.

[14] C. K. Leonard, A.R. Newton. "An Estimation Technique to Guide Low Power Resynthesis Algo-

rithms." Proceedings of the International Symposium on Low Power Design, pages 227-232, April 1995.

[15] D. Liu and C. Svensson. Trading speed for low power by choice of supply and threshold voltages. IEEE Journal of Solid State Circuits, 28(1):10–17, January 1993.

[16] A. A. Malik, R. K. Brayton, A. R. Newton, and A. L. Sangiovanni-Vincentelli. A modified approach to two-level logic minimization. In Proceedings of the IEEE International Conference on Computer Aided Design, Nov. 1988.

[17] P. McGeer and R. K. Brayton. "Consistency and Observability Invariance in Multi-Level Logic Synthesis" In IEEE International Conference on Computer-Aided Design, 1989.

[18] R. Marculescu, D. Marculescu, M. Pedram. Logic level power estimation considering spatiotemporal correlations. ICCAD'94, pages 294-299, 1994.

[19] S. Muroga, Y. Kambayashi, H.C. Lai, and J.N. Culliney. "The Transduction Method - Design of Logic Networks Based on Permissible Functions." In IEEE Transactions on Computers, October 1989.

[20] R. Rudell. "Logic Synthesis for VLSI Design," Ph.D. thesis, University of California, Berkeley, 1989.

[21] H. Savoj. Don't Cares in Multi-Level Network Optimization. PhD thesis, University of California, Berkeley, 1992.

[22] A. A. Shen, A. Ghosh, S. Devadas abd K. Jeutzer. "On average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks." In Proceedings of the International Conference on Computer-Aided Design, November 1992.

[23] V. Tiwari, S. Malik, P. Ashar. "Guarded Evaluation: Pushing Power management to Logic Synthesis/Design." In proceeding of the Symposium on Low Power Design, pages 221-226, April 1995.

[24] A. Wang. Algorithms for Multi-Level Logic Optimizations. Ph.D. Thesis, UC Berkeley 1989.