Multi-Level Network Optimization Targeting Low Power

Sasan Iman and Massoud Pedram

CENG Technical Report 94-06

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

March 1994

Multi-Level Network Optimization Targeting Low Power

Sasan Iman, Massoud Pedram
March, 1994

Abstract

This report describes a procedure for minimizing the power consumption in a boolean network under the zero delay model assumption. Power minimization is achieved by modifying the function of internal nodes of the network such that this modification does not result in an increase in the power consumption of nodes which have already been optimized. We first present a formal analysis of how changes in the switching activity of an internal node affects the switching activity of other nodes in the network. Using this analysis, we propose a new procedure for calculating power compatible don't cares in the network. We then present a new approach in using the power compatible don't care to optimize the function of the node such its power contribution after technology mapping is minimized. These procedures have been implemented and preliminary results show an average of 6% improvement in total power consumption of the network compared to results generated by full_simplify.

Content: **Page**

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, the power consumption is becoming the limiting factor on 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. For example, at the algorithmic level, correct data representation and choice of algorithms may significantly reduce power consumption [4]. At the system level, inactive hardware modules may be automatically turned off to save power; modules may be provided with the optimum supply voltage and interfaced by means of level converters. At the architectural design level, concurrency increasing transformations such as loop unrolling, pipelining and control flow optimization as well as critical path reducing transformations such as height minimization, retiming and pipelining may be used to allow a reduction in supply voltage without degrading system throughput [6][5]; Algorithm-specific instruction sets may be utilized that boost code density and minimize switching; A Gray code addressing scheme can be used to reduce the number of bit changes on the address bus [20]; Internal busses may be replaced by point-to-point connections to avoid driving a large number of modules on every bus access; Bus architectures with sub 1-volt swing and low standby current can be used [14]; On-chip cache may be added to minimize external memory references; Locality of reference may be exploited to avoid accessing global resources such as memories, busses or ALU's; Control signals that are "don't cares" can be held constant to avoid initiating nonproductive switching. At the device level, threshold voltage of MOS transistors can be reduced to match the reduced supply voltage [19][9]; Very low threshold voltages may be made possible by electrically controlling the threshold values (against process and temperature variations) by substrate modulation [2].

Once these system level, architectural and technological choices are made, it is the switching activity of the logic (weighted by the capacitive loading) that determines the power consumption of a circuit. In this paper, we will describe new techniques for power minimization at the technology independent phase of logic synthesis.

In order to minimize the power consumption of a network, we should minimize the contribution of each node to the total power consumption of the network. Network don't cares can be used to modify the function of each node to achieve this goal.

The rest of this report is organized as follows. In Section 2. we review the model used for power estimation. In Section 3. we discuss previous work on using don't cares to minimize the network area and power consumption. In Section 4. we present a formal analysis on the effect of local signal probabilities on the signal probability of other nodes in the network. In Section 5. we discuss how this information can be used to optimize nodes in the network. Experimental results and concluding remarks are presented in Sections 6. and 7.

2. Power Estimation

The dynamic power consumption in a boolean network is composed of two components. First is the power consumption due to steady-state transitions at the outputs of internal nodes in the network; The second component is due to hazard/glitches at the outputs.

The power consumption due to steady-state output changes is calculated using the following equation:

$$P_i = 0.5 \cdot V_d^2 \cdot f \cdot \sum_j c_i \cdot E_i$$

where V_d is the power supply voltage, f is the input clock frequency, c_i is the load seen by the node and E_i is the switching activity of node i.

In this report we assume that the primary input signals are spatially and temporally uncorrelated. We also assume a zero (non-glitch) delay model. Given these assumptions we can estimate the switching activity of each signal by the following equation:

$$E_i = 2 \cdot p_i \cdot (1 - p_i)$$

Given signal probabilities for the primary inputs of a boolean network, the signal probability p_i of node n_i is computed by building the global BDD for n_i and then applying the procedure presented in [13].

3. Previous Work

Network don't cares can be used for minimization of nodes in a boolean network [1]. Once the compatible don't cares [12] are computed for nodes in a network, each node can be optimized for area without any concern that changes in the function of this node might affect the function of primary outputs of the network. In [16] a procedure is presented where observability don't cares [3] and image projection techniques are used to compute the compatible local don't cares for each node in the network. The compatible local don't care for node n_i is then used to minimize the number of literals in the function of n_i.

The following procedure presented in [16] is used to compute the observability don't care (ODC) for node g in a boolean network:

$$ODC_g = \overline{\frac{\partial fo_i}{\partial g}} + \prod_{fo_i \in \mathrm{fanout} of g} \left(ODC_{fo_i} \right)$$

The procedure for minimizing the area of a boolean network is modified in [18] to minimize the power consumption of the network. The difference between this procedure and the procedure for minimizing the network area is in the cost function used for minimizing each node. Given the function for a node n_i and its local don't care, the following cost function is used to minimize the power consumption of node n_i:

$$c \cdot m \cdot \sum_{n_j \in \, cubes\,(n_i)} E_j + \sum_{n_k \in \, fanins\,(n_i)} E_k$$

where c is the fount load, m is the number of product terms in the cover, p_j is the local signal probability of *j-th* cube in the cover and p_k is the signal probability of *k-th* local input. The first term in this function approximates the power consumed by each cube of the function where m is used to approximate the penalty in having a large number of cubes. The second term is used to approximate the power due to the load capacitances of the immediate fanin nodes.

This method suffers from two shortcomings. First is that the given procedure does not consider how changes in the function of an internal node affects the signal probability of nodes in its transitive fanout. In general by changing the function of an internal node, it is possible to change the signal probability of nodes in its transitive fanout such that the power due to these fanout nodes is increased. For example if by modifying an internal node function, 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 reducing the switching activity of an internal node might result in an increase in the total power consumption of a network.

Example:
Assume that $p(a) = p(b) = 0.5$, $(f = a + g)$, $(dc_f = a.b)$ and $(g = a.b)$.
Thus $(dc_g = dc_f + a = a + b)$.

Before optimization:

$f = a + \bar{a}.b \Rightarrow p(f) = 3/4 \Rightarrow E(f) = 3/8 \Rightarrow P(f) = 15/8,$
$g = \bar{a}.b \Rightarrow p(g) = 1/4 \Rightarrow E(f) = 3/8 \Rightarrow C(g) = 3/8.$

Total power = $18/8$.

After optimization:

f is not changed to keep *P(f)* low.
set $g = 0$.
$f = a \Rightarrow p(f) = 1/2 \Rightarrow E(f) = 1/2 \Rightarrow P(f) = 5/2,$
$g = 0 \Rightarrow p(g) = 0 \Rightarrow E(f) = 0 \Rightarrow P(g) = 0.$

Total power = $5/2$.

$\square$

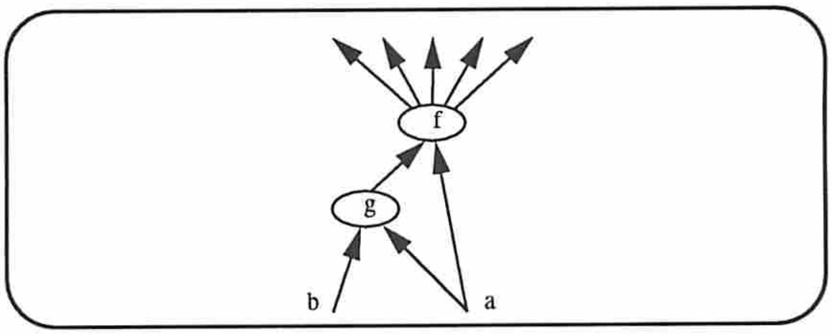

Figure 1

The second drawback is in the cost function used for optimizing nodes as this cost function does not clearly reflect the power cost of the two-level cover. When targeting two-level cover, the following cost function is more appropriate:

$$\sum_{n_j \in fanins(n_i)} SL(n_j, n_i) \cdot E_j + \sum_{c \in cubes(n_i)} E_c + \sum_{n_k \in fanouts(n_i)} SL(n_i, n_k) \cdot E_i$$

where $SL(n_j, n_i)$ is the number of times variable n_j appears in the sum-of-products expression of node n_i (in true and complemented form) Note that the capacitive loading of all lines is taken to be equal to some constant C_0 which drops out of the equation. Our experiments show that even this is not a good cost function to use during node minimization in a multi-level network.

In order to accurately estimate the power consumption at the output a node n_i, we need to use a load value c_i that more accurately reflects the load seen by the node **after technology mapping**. Given a node n_i and fanin node n_j, we define the factored load $FL(n_j, n_i)$ as the number of times variable n_j is used (in positive or negative form) in the factored form expression of node n_i. We can thereby calculate the power contribution of a node n_i (excluding its internal power consumption) by the following equation:

$$P_i = \sum_{n_j \in fanins(n_i)} FL(n_j, n_i) \cdot E_j + \sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) \cdot E_i$$

In this paper we use this equation to estimate the power contribution ***power_cost(n_i)*** of each node after the network is mapped. Experimental results show that the power estimate computed before technology mapping, using this approach closely matches the actual power consumption after technology mapping (see Section 6.)

A more accurate estimate of the power contribution of a node can be obtained by augmenting P_i with a weighted factor of the number of literals in the node $(A)_i$. Then *power_cost*$(n_i) = P_i + \alpha . A_i$ where α is the weighting coefficient.

4. Power Don't Care Analysis

The procedure for don't care calculation as described in [16] guarantees that the function of the network primary outputs will not change for any condition not specified in the external don't cares. However as the don't care for an internal node n_i is being computed, it is possible to change the global function of nodes n_j in the transitive fanout of n_i. The change in the function of nodes in the transitive fanout as the node is being optimized, is not of concern when area is being minimized since the change in each function will be within the observability don't care calculated for that node. However this observation does not apply to power minimization as power depends on signal probability of all nodes in the network. In the following a formal analysis of the relation between power and don't care calculation for tree networks is presented. This analysis is then extended to general networks.

4.1. Power Don't Care Analysis for Tree Networks

The following don't care relations for tree networks will be used in the following discussion:

Consider nodes f and g where function of node f is expressed in terms of variable g(Figure 2).

By definition: $ODC^f_g = \overline{\partial f/\partial g}$,
and $ODC_g = ODC^f g + ODC_f$
$\Rightarrow ODC^f g \subseteq ODC_g$.

From this definition node f is always more observable than node g:

$ODC_f \subseteq ODC_g$.

Also note that $g \cap ODC_g \neq \emptyset$.

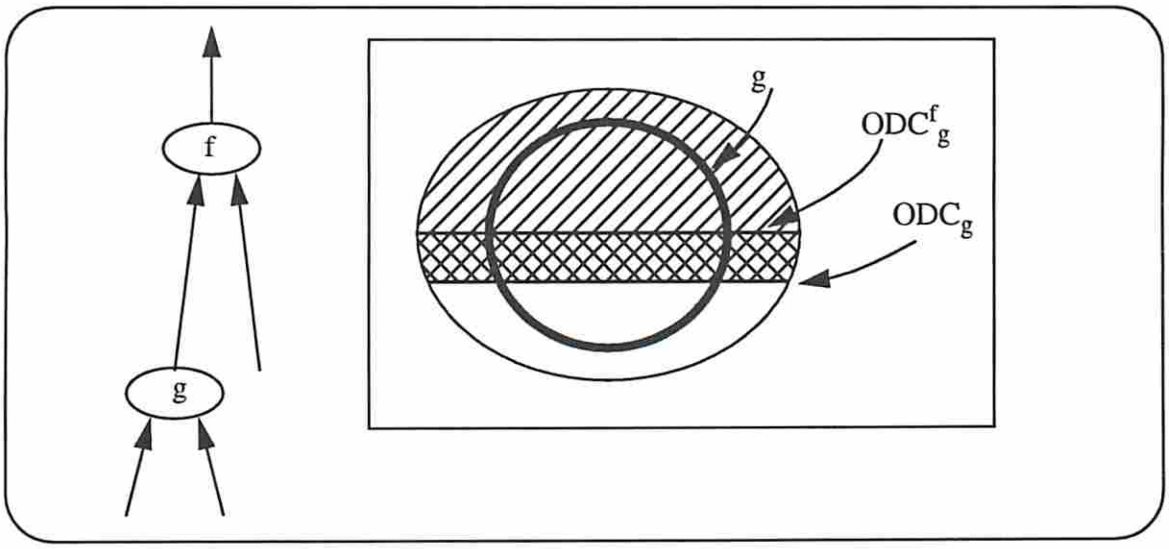

Figure 2

We can write the following relations for f and g where f_g is the cofactor of f with respect to variable g.

f_g: global conditions where $g = 1$ and $f = 1$
$f_{\bar{g}}$: global conditions where $g = 0$ and $f = 1$
$\bar{f}_g$: global conditions where $g = 1$ and $f = 0$
$\bar{f}_{\bar{g}}$: global conditions where $g = 0$ and $f = 0$

Now define the following:

$$\partial f^+/\partial g = f_g \cdot \bar{f}_{\bar{g}} \qquad\qquad \partial f^-/\partial g = f_g \cdot f_g \, .$$

By definition $\partial f^+/\partial g$ gives all global conditions for which the values for both f and g are the same while $\partial f^-/\partial g$ gives all global conditions for which f and g evaluate to opposite values, that is:

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

Note that $\partial f/\partial g = \partial f^+/\partial g + \partial f^-/\partial g$ which is the difference equation.

Figure 3 shows how the global space of the primary inputs is partitioned with respect to global functions 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 g will also be included in f. This means that by including these points in g, the number of points 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 g will be removed from f. This means that including these points in g will reduce the number of points in the onset of f (hence the down arrow).

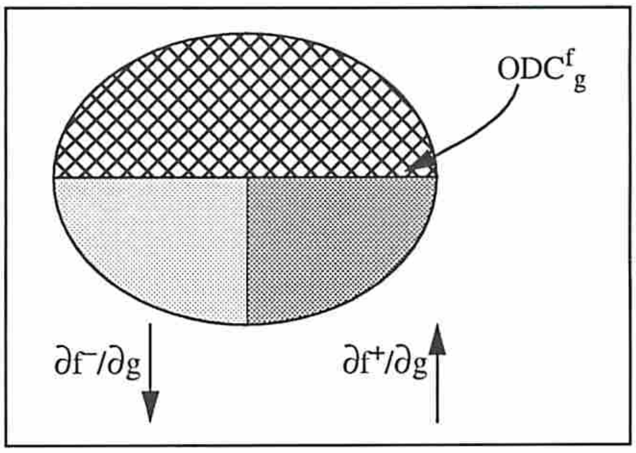

Figure 3

4.2. Don't Care Regions

We need to analyze the relationship between $\partial f^-/\partial g$ and $\partial f^+/\partial g$ and the global function of node g. Figure 4 shows this relation where points inside the inner circle are on-set points of function g and $ODC^f g \subseteq ODC_g$.

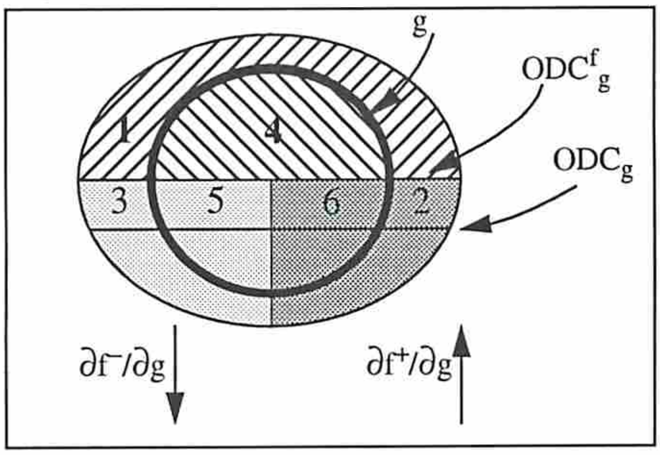

Figure 4

Don't care conditions for node g (region above line ODC_g) can be partitioned into six regions described below where v_i is a minterm in region i.

$g\,f$
$\uparrow$ O : region 1: including v_1 in g will not change f,
$\uparrow\uparrow$: region 2: including v_2 in g will increase $p(f)$ by $p(v_2)$,
$\uparrow\downarrow$: region 3: including v_3 in g will decrease $p(f)$ by $p(v_3)$,
$\downarrow$ O : region 4: removing v_4 in g will not change f,
$\downarrow\uparrow$: region 5: removing v_5 in g will increase $p(f)$ by $p(v_5)$,
$\downarrow\downarrow$: region 6: removing v_6 in g will decrease $p(f)$ by $p(v_6)$.

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. For example all minterms in region 2 are in the off-set of function g. We also know that region 2 is a subset of $\partial f^+/\partial g$ which means that both functions f and g will evaluate to the same value for minterms in this region. Hence we know that for all minterms in region 2, function f evaluates to zero. Now if we use region 2 in optimizing node g, we can only bring points in this region into the on-set of function g and hence the same points will be brought into the on-set of function f. This means that for each minterm v in region 2 which is brought into the on-set of function g, the probability of function f is increased by $p(v)$. Using the same line of reasoning, the exact change in the signal probability of node f can be calculated as minterms in other don't care regions of g are either included in or excluded from the on-set of g.

4.3. Using Don't Care Regions in Node Optimization

In a tree network, most nodes have more than one node in their transitive fanouts. This means that while optimizing a node n_i, it is necessary to consider the effect of changes in the function of n_i on all nodes in its transitive fanout. For a node n_i with k nodes in its transitive fanout, the number of don't care regions is given by $4k+2$. Figure 5 shows an example where node g has 2 nodes in its transitive fanout. As shown in the figure, 10 regions can be identified in ODC_g.

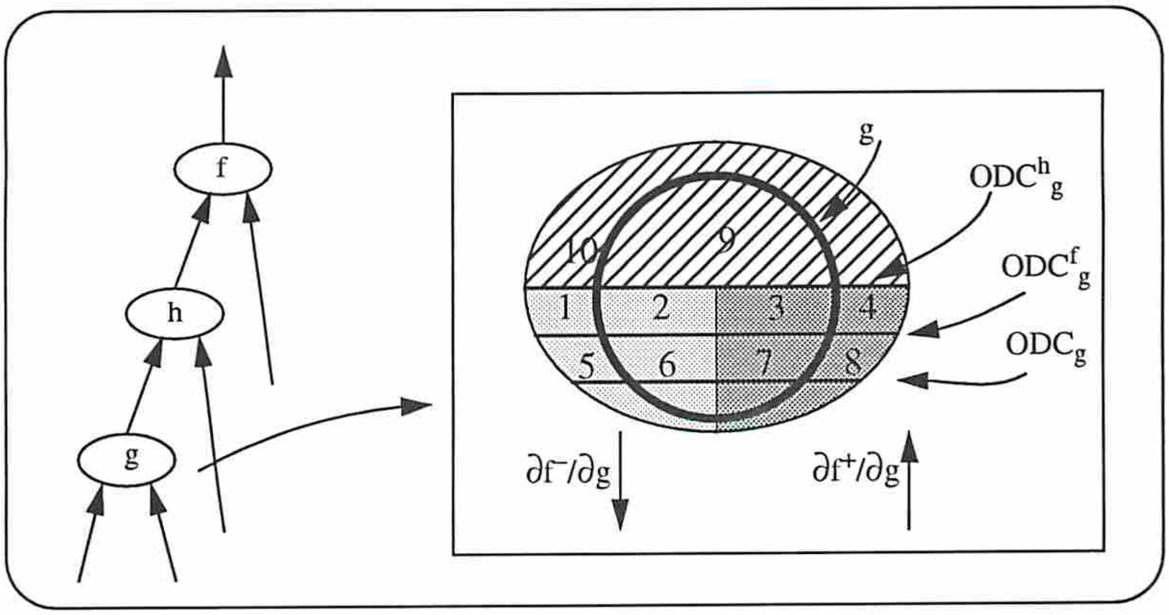

Figure 5

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 in the signal probability of these fanout nodes. This information, however, cannot efficiently be used during node optimization. In order to minimize the power consumption of node g, we need to minimize the switching activity of g and all its transitive fanout nodes. This would be very difficult as a case-by-case analysis of each don't care region is required.

There are other 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 on all don't care regions corresponding to its transitive fanout nodes (e.g. those shown in Figure 5) and 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 f can be made while optimizing different nodes in its transitive fanin. 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.

The complexity of power optimization procedure can be reduced if a decision is made as to increase or decrease the signal probability of a function after it has been optimized. This means that in later node optimizations, alternating decisions can not be made regarding the new signal probability of this node.

The following two theorems can be used to reduce the complexity of the power optimization procedure.

Theorem 1:

Given nodes *f* and *g* as shown in Figure 4, regions 3 and 6 are the empty sets and regions 2 and 5 are maximal if ODC_g is expressed as:

$$ODC_g = \bar{f} . ODC_f + ODC^f_g .$$

Proof: By definition regions 2, 3, 5 and 6 can be computed by the following equations:

$$\begin{aligned}
\text{Region } 2 &= \bar{g} . f_g . \bar{f}_{\bar{g}} . ODC_g,\\
\text{Region } 3 &= \bar{g} . f_{\bar{g}} . \bar{f}_g . ODC_g,\\
\text{Region } 5 &= g . f_{\bar{g}} . \bar{f}_g . ODC_g,\\
\text{Region } 6 &= g . f_g . \bar{f}_{\bar{g}} . ODC_g.
\end{aligned}$$

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

$$\begin{aligned}
\text{Region } 2 &= \bar{g} . f_g . \bar{f}_{\bar{g}} . ODC_f,\\
\text{Region } 3 &= \bar{g} . f_{\bar{g}} . \bar{f}_g . ODC_f,\\
\text{Region } 5 &= g . f_{\bar{g}} . \bar{f}_g . ODC_f,\\
\text{Region } 6 &= g . f_g . \bar{f}_{\bar{g}} . ODC_f.
\end{aligned}$$

Now if we instead substitute $\bar{\mathrm{f}} . \mathrm{ODC}_\mathrm{f} + \mathrm{ODC}^\mathrm{f}_\mathrm{g}$ for ODC_g and use ($\mathrm{f}=g . f_g + \bar{g} . f_{\bar{g}}$):

$$\begin{aligned}
\text{Region } 2 &= \bar{g} . f_g . \bar{f}_{\bar{g}} . ODC_f,\\
\text{Region } 3 &= \emptyset,\\
\text{Region } 5 &= g . f_{\bar{g}} . \bar{f}_g . ODC_f,\\
\text{Region } 6 &= \emptyset.
\end{aligned}$$

❑

Theorem 2:

Given nodes f and g as shown in Figure 4, regions 2 and 5 are the empty sets and regions 3 and 6 are maximal if ODC_g is expressed as:

$$ODC_g = f . ODC_f + ODC^f_g .$$

Proof: The 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 more than or the same as its current value. This is, for example, desirable when the signal probability of *f* after it is optimized, is more than 0.5 This will disallows any increase in the switching activity of the node beyond its present value. Since a decision is made to only allow an increase in the signal probability of function *f*, we only use regions 1, 2, 4 and 5 while optimizing a node *g* in the transitive fanin of f. Using theorem 1 we can compute ODC_g such that regions 3 and 6 are empty sets and regions 2 and 5 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 its effect on switching activity of node f.

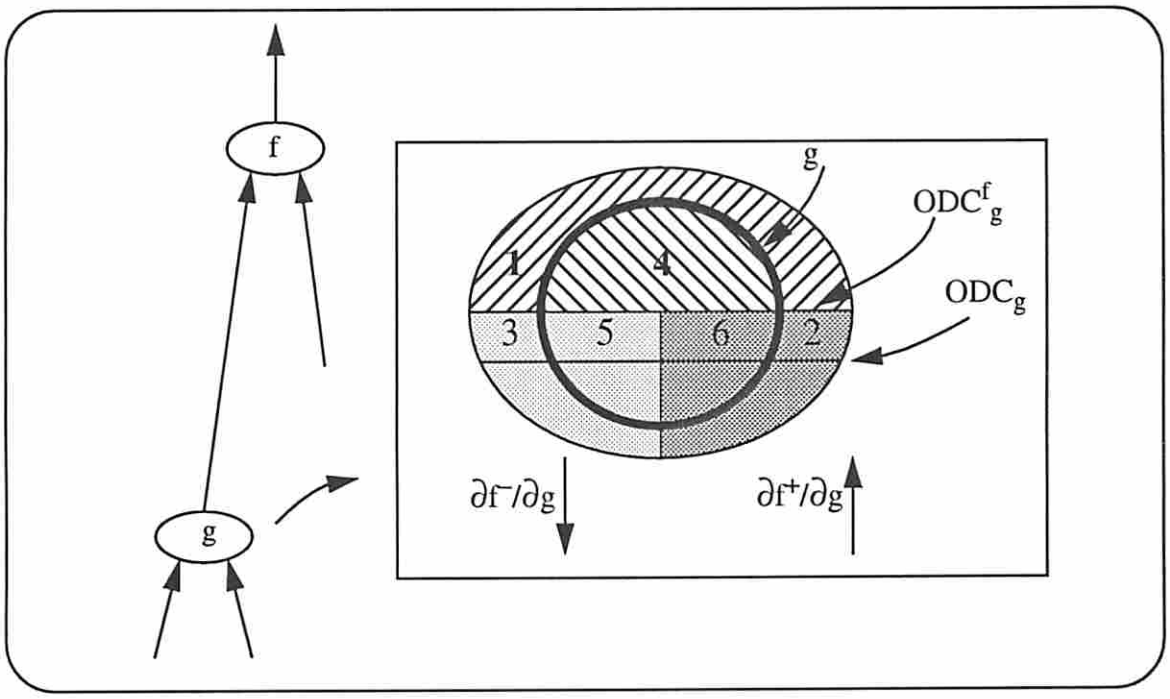

Figure 6

The same argument is used to apply theorem 2 when we want to disallow an increase in the signal probability of node f (if the signal probability of node is less than 0.5 after it is optimized).

Theorems 1 and 2 are thus used as a don't care filter while calculating the ODC$_g$ for a node g in the network. The power compatible don't care for a node g with fanout f can then be calculated by using the following modified procedure:

$$ODC_g = ODC^f_g + PODC_f$$

Once ODC_g is used to compute the new function for node g, $PODC_g$ is calculated using the following procedure. $PODC_g$ is then stored at node g for future reference.

Example. consider example 1.
Assume $(f = a + g)$, $(dc_f = \bar{a}.b)$ and $(g = \bar{a}.b)$.

Before optimization:

$$f = a + \bar{a}.b \Rightarrow p(f) = 3/4 \Rightarrow E(f) = 3/8 \Rightarrow P(f) = 15/8$$
$$g = \bar{a}.b \Rightarrow p(g) = 1/4 \Rightarrow E(f) = 3/8 \Rightarrow P(g) = 3/8$$

After optimization:

$$p(f) = 3/4 > 1 \Rightarrow ODCg = \bar{f}.\ ODC_f + ODC^f_g$$
$$\Rightarrow ODCg = a.$$

set $g = \overline{a}.b$ (don't change)

$f = a + \overline{a}.b \Rightarrow p(f) = 3/4 \Rightarrow E(f) = 3/8 \Rightarrow P(f) = 15/8$
$g = \overline{a}.b \Rightarrow p(g) = 1/4 \Rightarrow E(f) = 3/8 \Rightarrow P(g) = 3/8.$

❑

```
function find_power_odc(g, odc)
g is the node function and odc is the observability don't care;
begin
    if (p(g) > 0.5) then
            $PODC_g = \overline{g} . ODC_g$.
    else
            $PODC_g = g . ODC_g$.
    end if
end
```

Figure 7

Using the new method, a new ODC_g is calculated. The new don't care for g is not used in changing the function of g since the transition density of g would only increase.

The procedure presented above guarantees that after network optimization the switching activity of each node in the network is less than or equal to its switching activity right after if was optimized. This however 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 optimized.

4.4. Analysis for General Networks

The approach used to analyze a tree network can be directly used to analyze a general boolean network. However since the observability don't care relations for trees do not hold in a DAG, the number of don't care regions for a node is much larger than the number of don't cares for trees[1]. The number of don't care regions for a node g in a DAG is given by $2(3^n + 1)$ where n is the number of transitive fanout nodes of g.

The power compatible don't care is also calculated by using the following equation.

$$ODC_g = \overline{\frac{\partial fo_i}{\partial g}} + \prod_{fo_i \in \text{fanout} of g} \left(PODC_{fo_i} \right)$$

Once the *ODC* is computed for node g, *PODC* can be computed using procedure *find_power_odc*. The power compatible don't care $PODC_g$ for a node g can then be used to optimize the

1. It is therefore impractical to use the don't care regions for network optimization except for small circuits. The filtering scheme proposed by Theorems 1 and 2 can however be easily applied.

function of node g with the knowledge that any use of this don't care will only result in an improvement of the current partial solution.

5. Node Optimization

In the past, node optimization using don't cares has been mainly used to minimize the area of boolean networks. Programs such as Espresso [15] and MINI [8] have been developed to optimize the two level representation of boolean functions by reducing the number of cubes in the cover of the function. However the cost functions used in these programs cannot directly be used to minimize the power consumed by the circuit implementing a given two level boolean equation. This means that a new procedure needs to be developed which targets minimizing the power consumption of the node rather than its area.

Given a node i with k local fanins in a multi-level boolean network with m primary inputs, the local don't care (*DC*) of each node is computed by computing the complement of the image of all care points in R^m onto R^k*[16]*. For power optimization purposes, *DC* can further be divided into Function Modifying Don't Care (*FMDC*) and Function Preserving Don't Care (*FPDC*). *FPDC* is the complement of the range of R^m onto R^k. *FMDC* is then given by *(DC - FPDC)*. While optimizing node i, we can freely use *FPDC* to optimize the function for area without changing the global function of the node.

We can obtain the minimum and maximum signal probability for node i by computing *(F-FMDC)* and *(F + FMDC)*. We can then select the function with minimum switching activity and then optimize that function with the *FPDC*. Our experiments however show that the increase in the power of the node due to the increase in its area will be more than the power gained by reducing the switching activity of the output. In fact more gain can be obtained by reducing the power consumption of the node and its immediate fanins as described shortly. The following example illustrates this adverse effect.

Example:

Assume ($f = b.\ c + b.\ c$) and ($dc = a.\ b.\ c.\ d$). *Then:*

$$p(f) = 1/2 \Rightarrow E(f) = 64/256$$

if we allow ($ff = f + dc$) then:

$$ff = \overline{b}.\ c + b.\ c + a.\ b.\ d$$

where

$$p(ff) = 9/16 \text{ and } E(ff) = 63/256.$$

which reduced the switching activity at the output of the node but the literal count is drastically increased.

❑

In calculating the total power consumption of a network, the switching activity of each node is multiplied by the load seen at the output. This means that if any fanin is removed from the function of a node, the load seen at the output of this fanin and hence the power consumption is reduced.

Given an incompletely specified function *ff*, it is often possible to implement *ff* using different variables as the support. A variable support is said to be minimal if it is not a proper subset of any other variable support. The set of minimal variable supports for a function contains all its minimal supports. 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 variable supports for node *F*.

In order to choose the best possible variable support for a node n, it is necessary to compute the set of minimal variable supports for n. A procedure presented in [7] can be used to compute the set of minimal variable supports for a function f. The difficulty with this method is that it requires a cover of the on-set and off-set of the function. 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. A more efficient method [10] uses reduced off-sets [11] to compute the set of minimal variable supports of a function f.

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 in implementing the function. A simple cost function 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 can be computed by dropping variables not in the support [7].

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 $dc_{reduced}$. Figure 9.a shows the on-set and don't care for a function *f*. Figure 9.b shows the don't care assignment which is used to eliminate variable *x* from the support to obtain f_{new} and Figure 9.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 can be removed from the on-set of the function *f*.

Given a cube *v* representing the variables removed from the on-set of a function *f* and don't care for function *f*, the reduced don't care for *f* is given by $C_v(dc)$ where $C_v(dc) = dc_v.dc_{\bar{v}}$.

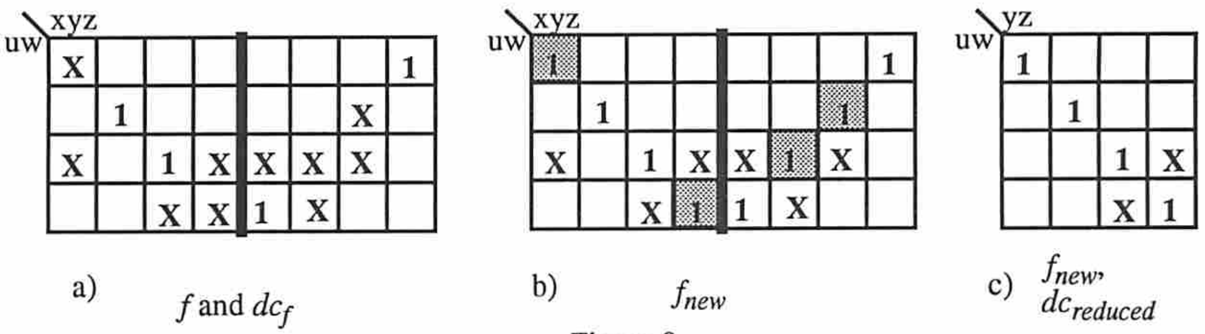

Figure 8

Given a function *f*, its don't care *dc* and a variable support *v*, the following procedure is used to optimize the power consumption of the function.

```
function node_fanin_optimize(f, dc, v)
f is the function and dc is the don't care of node, v is a variable support;
begin
    p= cube representing eliminated variables.
    $f_{reduced}$ = function_elim_variables(f, p).
    $dc_{reduced} = C_p(dc)$.
    $f_{new}$ = espresso($f_{reduced}$, $dc_{reduced}$).
    return $f_{new}$.
end
```

Figure 9

The given procedure will provide a low area implementation which has the lowest sum of switching activity on the immediate fanins of the node. However it is 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 as presented in Section 2., 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 following procedure is then used to select the lowest power cost implementation of the function:

```
function node_power_optimize(f, dc)
f is the function and dc is the don't care of node;
begin
    $f_{esp}$ = espresso(f, dc).
    V = find_k_minimal_switching_activity_sup(f, dc).
    for each $v \in V$ do
        $f_{tmp}$ = node_fanin_optimize(f, dc, v)
        if power_cost($f_{tmp}$) < power_cost($f_{esp}$) then
            $f_{esp} = f_{tmp}$
        endif
    endfor
    return $f_{new}$.
end
```

Figure 10

The following procedure is used for minimizing the power consumption of a boolean network:

```
function power_ optimize(G)
G(V, E) is a DAG representing a boolean network;
begin
    for each node $n \in G$ in reverse depth first search order do
            Calculate $ODC_n$.
            $f_{new} = node\_pow\_optimize(f_n, ODC_n)$.
            $PODC_n$ = find_power_odc($f_{new}$, $ODC_n$)
            store $PODC_n$ at node $n$ for computation of $ODC_j$ where $j$ is fanin of $n$.
    endfor
```

Figure 11

6. Results

The procedures presented in this paper were implemented in a program called *power_-full_simplify* and the result were compared to those of the *full_simplify* command in the *SIS* package. Table 1 summarizes the results of the optimization obtained after running the *script.rugged* script. Column 3 shows the estimated power for the internal nodes of the network after the optimization process. Column 4 shows the power estimate for the primary inputs of the network after the optimization. After the optimization, circuits were mapped and the network power was calculated under a zero delay model. Column 7 shows the ratio of the sum of internal power and input power to the power after technology mapping. The data in column 7 shows that the factored form power estimates used in this paper are highly correlated to the power consumption of the technology mapped network.The results in Table 2 are generated by replacing the full_simplify command in *script.rugged* by *power_full_simplify* command. All results are normalized with respect to results in Table 1. Column 7 is the ratio of the sum of internal and input power before mapping to the circuit power after mapping. This data again shows the accuracy of our power estimation method before technology mapping. As results show, on average we were able to obtain 6% improvement in terms of power and 3% improvement in terms of area. This was accomplished mainly by restructuring the network such that load on nodes with high switching activity is minimized. We expect to obtain better results if external don't cares are given for the circuits being optimized. The circuits we used had no external don't cares, as a result the *ODC* for these networks were usually small.

	Pre-Mapping			Post-Mapping		
example	lits (fact)	internal power estimate	input power estimate	active area	power	estimation ratio x100
9symml	184	18.29	25.75	157	2.85	1.54
apex6	745	62.19	106.00	633	10.2	1.65
apex7	244	21.34	37.00	214	3.58	1.63
b9	122	9.34	22.00	110	1.83	1.72
bw	160	25.36	14.00	136	2.24	1.75
f51m	91	11.94	11.25	83	1.52	1.53
k2	1135	47.62	66.50	1005	6.40	1.78
lal	104	6.58	16.75	88	1.31	1.79
misex1	52	5.25	8.00	46	0.79	1.67
misex2	103	4.25	16.00	93	1.16	1.75
pm1	49	4.05	7.75	46	0.68	1.74
rot	671	69.02	98.25	592	9.82	1.70
sao2	129	8.62	20.25	117	1.90	1.52
squar5	56	6.77	8.25	49	0.93	1.61
term1	168	13.77	23.00	145	2.17	1.69
ttt2	215	20.26	30.75	187	3.08	1.66
x1	302	20.27	50.75	274	4.32	1.64
x4	379	35.27	55.00	336	5.41	1.67
Average						1.67

Table 1:

	Pre-Mapping			Post-Mapping		
example	lits (fact)	internal power estimate	input power estimate	active area	power	estimation ratio
9symml	1.21	0.48	1.64	0.81	0.91	1.97
apex6	0.99	0.99	0.96	0.98	0.97	1.65
apex7	1.00	1.06	0.94	1.00	0.95	1.68
b9	1.00	1.09	0.91	1.04	1.00	1.66
bw	0.96	0.99	0.86	0.97	0.94	1.76
f51m	1.12	0.84	1.04	0.93	0.89	1.61
k2	1.00	1.07	0.78	0.97	0.88	1.83
lal	1.00	1.04	0.97	0.99	0.97	1.82
misex1	0.96	1.11	0.78	0.93	0.92	1.66
misex2	1.01	1.16	0.83	0.99	0.90	1.75
pm1	1.00	1.17	0.84	1.00	0.93	1.78
rot	0.99	1.04	0.94	0.98	0.98	1.70
sao2	1.01	1.18	0.85	0.97	0.94	1.53
squar5	0.93	0.87	0.94	0.92	0.90	1.62
term1	1.06	0.89	1.03	1.00	0.98	1.68
ttt2	0.99	1.02	0.91	0.98	0.95	1.67
x1	1.00	1.08	0.95	1.00	1.00	1.63
x4	1.00	1.00	1.00	0.99	0.99	1.68
Average	1.01	1.00	0.95	0.97	0.94	1.70

Table 2:

7. Concluding Remarks

In this report a method is presented which allows us to minimize the power consumption of a network. Using the techniques presented here it is possible to guarantee that local node optimizations will not increase the power consumption in the transitive fanout nodes. This means that local nodes can be optimized without concerns on how changes in the function of the current node affect the power consumption in the rest of the network. Future work includes the development of low power equivalent of other commands in the SIS rugged script.

8. References

[1] [bar88] K. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. Multi-level logic minimization using implicit don't cares. In IEEE *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 7, pages 723–740, June 1988.

[2] [Burr-93] J. B. Burr. Stanford ultra low power CMOS. In *Proceedings of Hot Chips Symposium V*, pages 583–588,

June 1993.

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

[4] [Chandra-cicc94] A. P. Chandrakasan, R. Allmon, A. Stratakos, and R. W. Broderson. Design of portable systems. In Proceedings of the IEEE Custom Integrated Circuits Conference, May 1994.

[5] [hyper-lp] A. P. Chandrakasan, M. Potkonjak, J. Rabaey, and R. W. Broderson. HYPER-LP: A system for power minimization using architectural transformation. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, pages 300–303, November 1992.

[6] [Chandra-jssc92] 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] [halatsis-gaitanis] 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.

[8] [mini] S. 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.

[9] [Liu-jssc93] 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.

[10] [unicut] S. Iman, M. Pedram, C. Fabian and J. Cong. Finding uni-directional cuts based on physical partitioning and logicrestructuring. In *Proc. the 4th ACM/IEEE Physical Design Workshop*, April 1993.

[11] [reduced-offset] Abdul A. Malik, Robert K. Brayton, A. Richard Newton, and Alberto L. Sangiovanni-Vincentelli. A modified approach to two-level logic minimization. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, Nov. 1988.

[12] [Muroga-89-transduction] 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*, 1989.

[13] [Najm-91] F. Najm. Transition density, a stochastic measure of activity in digital circuits. In *Proceedings of the 28th Design Automation Conference*, pages 644–649, June 1991.

[14] [Nakagome-jssc93] Y. Nakagome, K. Itoh, M. Isoda, K. Takeuchi, and M. Aoki. A sub-1-V swing bus architecture for future low power ULSIs. IEEE *Journal of Solid State Circuits*, 28(4):414–419, April 1993.

[15] [Rudell-85] R. Rudell and A. Sangiovanni-Vincentelli. Espresso-mv: Algorithms for multiple-valued logic minimization. In *Proceedings of the* IEEE *Custom Integrated Circuits Conference*, pages 230–234, May 1985.

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

[17] [savoj-mcnc] H. Savoj, R. K. Brayton, and H. J. Touati. Use of image computation techniques in extracting local don't cares and network optimization. In *International Workshop on Logic Synthesis*, May 1991.

[18] [Shen-92] A. A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. On average power dissipation and random pattern testability of CMOS combinational logic networks. In *Proceedings of the* IEEE *International Conference on Computer Aided Design*, November 1992.

[19] [shimohigashi-93] K. Shimohigashi and K. Seki. Low-voltage ULSI design. IEEE *Journal of Solid State Circuits*, 28:408–413, April 1993.

[20] [Su-compcon-93] C-L. Su, C-Y. Tsui, and A. M. Despain. Low power architecture design and compilation techniques for high-performance processors. In *CompCon'94 Digest of Technical Papers*, pages 489–498, February 1994.