Multi-Level Network Optimization for Low Power*

Sasan Iman, Massoud Pedram

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089

Abstract

This paper describes a procedure for minimizing the power consumption in a boolean network under the zero delay model. Power is minimized by modifying the function of each intermediate node in the network such that the power consumption of the node is decreased without increasing the power consumption of the other nodes in the network. A formal analysis of how changes in the switching activity of an intermediate node affect the switching activity of other nodes in the network is given first. Using this analysis, a procedure for calculating the set of compatible power don't cares for each node in the network is presented. Finally it is shown how these don't cares are used to optimize the network for low power. These techniques have been implemented and results show an average of 10% improvement in total power consumption of the network compared to the results generated by the conventional network optimization techniques.

1.Introduction

Portability of modern digital applications places severe restrictions on the 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 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 a reduction in supply voltage without degrading system throughput [4]. At the device level, threshold voltage of MOS transistors can be reduced to match the reduced supply voltage [6]; 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. Network don't cares can be used to modify the function of each node to achieve this goal.

The rest of this paper 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 of the effect of local signal probabilities on the signal probability of other nodes in the network. In section 5. we discuss how this information is used to optimize nodes in the network. Experimental results and concluding remarks are presented in sections 6. and 7.

2.A power estimation model

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 intermediate nodes; The second component is due to hazard/glitches at the outputs.

The power consumption due to steady-state output changes is calculated using equation (1) where V_d is the 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.

$$P_i = 0.5 \cdot V_d^2 \cdot f \cdot \sum_j c_i \cdot E_i \qquad (1)$$

In this paper we assume that the primary input signals are spatially and temporally uncorrelated. We also assume a zero (non-glitch) delay model. Given these assumptions, the switching activity of node n_i is given by equation (2) where p_i is signal probability of the node:

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

Given signal probabilities for the primary inputs of a boolean network, p_i is computed using the global BDD for n_i [9].

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 are computed for all nodes, 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 [10] 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 nodes 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.

Equation (3) [10] is used to compute the observability don't care (ODC) for node g in a boolean network:

* This research was supported in part by the NSF's Research Initiation Award under contract no. MIP-9211668 and the Intel Corp.

$$ODC_g = \overline{\frac{\partial fo_i}{\partial g}} + \prod_{fo_i \in \text{fanout of} g} (ODC_{fo_i}) \quad (3)$$

The procedure for minimizing the area of a boolean network is modified in [11] 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:

$$\left(c \cdot m \cdot \sum_{n_j \in cubes(n_i)} E_j\right) + \left(\sum_{n_k \in fanins(n_i)} E_k\right) \quad (4)$$

In this equation c is the fount load, m is the number of product terms in the cover, E_j is the switching activity of *j-th* cube in the cover and E_k is the switching activity of *k-th* local input. The first term 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 account for the loading capacitances on the fanin nodes.

This method suffers from two shortcomings. First is that the proposed procedure does not consider how changes in the function of an intermediate node affect the signal probability of nodes in its transitive fanout. In general by changing the function of an intermediate 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. The following example demonstrates how reducing the switching activity of an intermediate node might result in an increase in the total power consumption of a network.

Example: Assume:

p(a) = p(b) = 0.5 and ($f = a + g$), ($dc_f = a.b$) and ($g = \bar{a}.b$).

Also assume f has 5 fanouts.Thus ($dc_g = dc_f + a = a + b$).

Before optimization:

$p(f) = 3/4 \Rightarrow E(f) = 3/8 \Rightarrow$ P(f) = 15/8,

$p(g) = 1/4 \Rightarrow E(f) = 3/8 \Rightarrow$ P(g) = 3/8. Total power = $18/8$.

After optimization:

f is not changed to keep $P(f)$ low and set $g = 0$. This means that f =a is the global function of f.

$p(f) = 1/2 \Rightarrow E(f) = 1/2 \Rightarrow$ P(f) = 5/2,

P(g) = 0 => E(f) = 0 => P(g) = 0. Total power = $5/2$.

❑

The second drawback is that the cost function used for optimizing nodes does not clearly reflect the power cost of the two-level cover.

In order to accurately estimate the power consumption at the output of 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:

$$P_i = \sum_{n_j \in fanins(n_i)} FL(n_j, n_i) \cdot E_j + \sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) \cdot E_i \quad (5)$$

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

In order to account for the internal power consumption of a node, one can augment 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 [10] 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 intermediate node n_i is being used to optimize n_i, it is possible to change the global function of nodes n_j in the transitive fanout of n_i. These changes are not important when area is being minimized since the change in the function of each fanout node will be within the observability don't care calculated for that node. However these changes may adversely affect the power consumption in the network.

4.1.Analysis for tree networks

Consider nodes f and g where function of node f is expressed in terms of variable g(Figure 1). By definition: $ODC^f_g = \overline{\partial f/\partial g}$, and $ODC_g = ODC^f_g + ODC_f \Rightarrow ODC^f_g \subseteq ODC_g$. Also note that $g \cap ODC_g \neq \emptyset$.

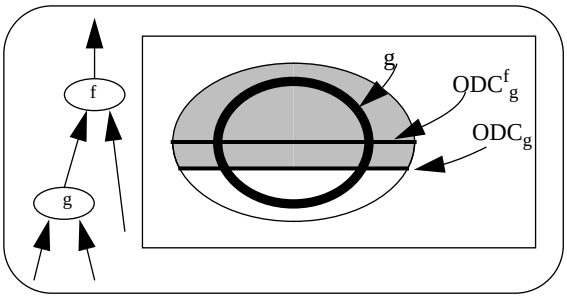


Figure 1

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:

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

From this definition $\partial f^+/\partial g$ gives all global conditions for which the values for both f and g evaluate the same and $\partial f^-/\partial g$

gives all global conditions for which f and g evaluate to opposite values. Note that $\partial f / \partial g = \partial f^{+} / \partial g + \partial f^{-} / \partial g$ which is the difference equation.

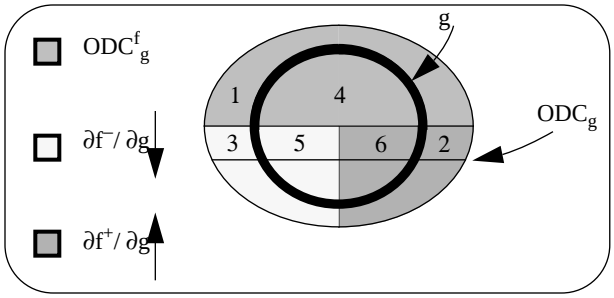


Figure 2

Figure 2 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. Region $\partial f^{-} / \partial g$ specifies all the points in the global space which if included in g will be removed from f. Figure 2 also shows the relationship between $\partial f^{-} / \partial g$ and $\partial f^{+} / \partial g$ and the global function of node g where points inside the inner circle are on-set points of function g and $ODC^{f}g \subseteq ODC_g$.

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

g f

↑ O : *region 1: including v_1 in g will not change f,*

↑ ↑ : *region 2: including v_2 in g will increase p(f) by p(v_2),*

↑ ↓ : *region 3: including v_3 in g will decrease p(f) by p(v_3),*

↓ O : *region 4: removing v_4 in g will not change f,*

↓ ↑ : *region 5: removing v_5 in g will increase p(f) by p(v_5),*

↓ ↓ : *region 6: removing v_6 in g will decrease p(f) by p(v_6).*

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 $4k+2$.

In order to minimize the contribution of node g to the power consumption of the network we need to minimize the switching activity of g and all its fanouts. 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 g in the signal probability of these fanout nodes. However 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 fanouts and this analysis 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 in 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 while optimizing nodes in its transitive fanin, alternating decisions cannot 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 (Figure 2), regions 3 and 6 are empty sets and regions 2 and 5 are maximal if ODC_g is expressed as:

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

Theorem 2:

Given nodes f and g (Figure 2), regions 2 and 5 are empty sets and regions 3 and 6 are maximal if ODC_g is expressed as:

$$ODC_g = f.\ ODC_f + ODC^{f}_{g}.$$

Theorems 1 and 2 can be used as follows. Assume that after optimizing a node f, we decide that as other nodes in the network are optimized we do not want the signal probability of this node to decrease below its current value. This is, for example, desirable when the signal probability of f after it is optimized, is more than 0.5. This aforesaid condition will thus disallow any increase in the switching activity of the node. Therefore we must only use don't care regions 1, 2, 4 and 5 in Figure 4 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. Theorem 2 is used when we do not want the signal probability of node f to increase above its current value as other nodes in the network are optimized.

Theorems 1 and 2 are thus used as don't care filters while calculating the ODC_g for a node g in the network. ODC_g for a node g with fanout f is calculated using Equation (6).

$$ODC_g = ODC^{f}_{g} + PODC_f \qquad (6)$$

After ODC_g is used to optimize the function of node g, $PODC_g$ is calculated using the procedure in Figure 3. $PODC_g$ is then stored at node g to be used in computing the don't care for fanin nodes of g.

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

Figure 3

The procedure presented above guarantees that after net-

work optimization the switching activity of each node in the network is less than or equal to its switching activity right after is 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.

4.2.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 regions for trees. Indeed 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 fanouts of g.

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

$$ODC_g = \overline{\frac{\partial fo_i}{\partial g}} + \prod_{fo_i \in \text{fanout of } fg} (PODC_{fo_i}) \qquad (7)$$

Once the *ODC* is computed for node *g*, *PODC* can be computed using the *find_power_odc* procedure. ODC_g as given in equation (7) can be used to optimize the 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

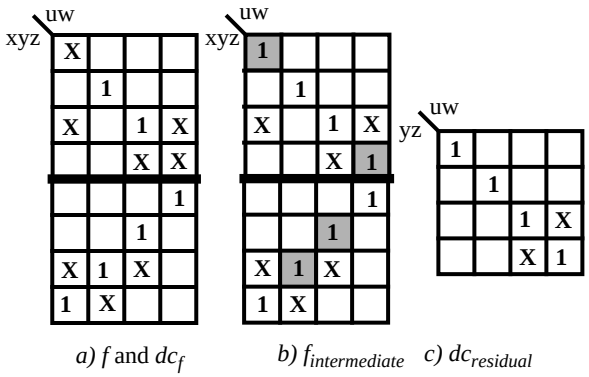
In the past, node optimization using don't cares has been mainly used to minimize the area of boolean networks. Programs such as Espresso and MINI have been developed to optimize the two level representation of boolean functions by reducing the number of cubes in the cover of the function. However the 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.

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 [5] is 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. This operation is computation expensive and the resulting off-set might have an exponential size. A more efficient method [7] uses reduced off-sets [8] 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 activities 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 [5].



a) f and dc_f *b)* $f_{intermediate}$ *c)* $dc_{residual}$

Figure 4

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 $f_{intermediate}$. 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 "*residual don't care*" $dc_{residual}$. Figure 4.a shows the on-set and don't care for a function *f*. Figure 4.b shows the don't care assignment that is used to eliminate variable *x* from the support to obtain $f_{intermediate}$ and Figure 4.c shows the residual don't care for function *f* after variable *x* is eliminated from the k-map. Using $dc_{residual}$ 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 $C_v(dc) = dc_v.dc_{\bar{v}}$.

This 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. Given a node function *f* and its don't care *dc*, the procedure in Figure 5 is used to select the lowest power cost implementation of *f*. The

procedure in Figure 6 is then used for minimizing the power consumption of a boolean network.

```
function node_power_optimize(f, dc)
    f is the function and dc is the don't care of node;
begin
    f_new = espresso(f, dc).
    V = find_k_min_switching_activity_sup(f, dc).
    for each v ∈ V do
        p= cube representing eliminated variables
        f_intermediate = function_elim_variables(f, p)
        dc_residual = C_p(dc)
        f_tmp = espresso(f_intermediate, dc_residual)
        if power_cost(f_tmp) < power_cost(f_new) then
            f_new = f_tmp
        endif
    endfor
    return f_new.
end
```

Figure 5

```
function
 optimize(G)
    G(V, E) is a DAG for a boolean network;
begin
    for node n ∈ G in reverse depth first order do
        Compute 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 to compute ODC_j
            where j is fanin of n
    endfor
```

Figure 6

6.Results

The procedures presented in this paper were implemented in a program called *power_full_simplify* and the results for both multi-level and two-level examples were compared to those of the *full_simplify* command in the *SIS* package.

Table 1. shows the results of the optimization after running the *script.rugged* script on two level examples. Column 1 gives the number of literals in the factored form., Columns 2 and 3 show the estimated power for the intermediate nodes and network primary inputs after the optimization process. Note that the sum of these two columns gives the total power estimate before mapping. Circuits were mapped and the network power was calculated under a zero delay model. Column 4 and 5 give the area and power of the mapped networks.

The results in Table 2: are generated by replacing the full_simplify command in the *script.rugged* with the *power_full_simplify* command. All results are normalized with respect to results in Table 1. As results show, on average we were able to obtain 11% improvement in terms of power and 7% improvement in terms of area.

Table 3. and Table 4. show the same analysis for multi-level examples. We were able to obtain 10% improvement in terms of power and 1% improvement in terms of area.

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.

7.Concluding Remarks

In this paper a method is presented that allows us to minimize the power consumption of a network. 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. Future work includes the development of low power equivalents of other technology independent logic operations.

8.References

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

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

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

[11] 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.

Table 1.full_simp for two-level examples

	Pre-Map			PostMap	
ex	1	2	3	4	5
5xp1	114	13.7	15.2	101	17.3
9sym	211	22.4	24.7	183	29.8
Z5xp1	116	14.7	14.5	105	18.0
Z9sym	202	21.5	23.7	179	28.8
b12	84	6.10	14.5	76.6	13.0
bw	160	25.4	14.0	136	21.2
clip	132	15.3	15.0	120	19.3
duke2	446	26.7	44.5	397	40.5
e64	253	1.25	32.7	293	20.8
inc	101	8.94	14.0	91.4	13.8
misex1	52	5.25	8.00	46.4	7.37
misex2	103	4.25	16.0	92.8	11.2
misex3c	451	45.4	60.0	379	61.8
rd84	145	18.3	13.2	131	20.4
sao2	129	8.62	20.2	117	18.8
spla	424	29.1	41.2	365	37.6
squar5	56	6.77	8.25	49.2	8.86
vg2	88	3.53	18.2	85.4	14.1

Table 2.power_full_simp for two-level examples

	Pre-Map			PostMap	
ex	1	2	3	4	5
5xp1	1.01	0.69	0.97	0.95	0.91
9sym	0.91	0.18	1.53	0.65	0.81
Z5xp1	0.97	0.67	0.79	0.87	0.83
Z9sym	1.00	1.02	0.98	0.99	1.01
b12	0.92	1.06	0.84	0.93	0.89
bw	0.96	0.99	0.86	0.97	0.93
clip	0.99	0.99	0.88	0.95	0.94
duke2	1.00	1.07	0.93	0.99	0.98
e64	1.00	1.33	0.51	0.80	0.34
inc	0.98	1.10	0.88	0.98	0.98
misex1	0.96	1.11	0.78	0.93	0.93
misex2	1.01	1.16	0.83	0.99	0.90
misex3c	1.01	1.02	1.00	1.02	1.01
rd84	0.92	0.87	0.91	0.84	0.85
sao2	0.99	1.13	0.86	0.98	0.94
spla.esp	0.98	1.12	0.80	1.02	0.97
squar5	0.89	0.88	0.82	0.86	0.84
vg2	0.99	1.00	0.97	0.99	0.97
Avg	0.97	0.97	0.90	0.93	0.89

Table 3.full_simp for multi-level examples

	Pre-Map			Post-Map	
ex	1	2	3	4	5
9symml	184	18.2	25.7	157	28.4
apex6	745	62.1	106	632	93.3
apex7	244	21.3	37.0	214	32.5
cm138a	31	0.81	5.25	24.1	3.01
cm42a	34	2.84	5.50	24.5	3.66
cm85a	46	1.57	9.25	41.7	6.58
decod	52	5.98	4.00	57.5	5.27
des	3483	469	261	2893	406
f51m	91	11.9	11.2	83.0	14.5
frg2	893	92.6	96.7	702	106
i6	458	55.4	54.0	423	63.8
i7	589	65.5	71.0	497	77.7
k2	1135	47.6	66.5	1005	62.0
lal	104	6.58	16.7	88.1	11.9
pm1	49	4.05	7.75	46.4	6.02
term1	168	13.8	23.0	145	21.4
ttt2	215	20.3	30.7	187	29.2
vda	615	40.4	26.2	532	38.3

Table 4.power_full_simp for Multi-level examples

	Pre-Map			Post-Map	
ex	1	2	3	4	5
9symml	1.20	0.56	1.56	0.79	0.90
apex6	0.99	0.99	0.96	0.98	0.97
apex7	1.00	1.06	0.94	1.00	0.95
cm138a	1.06	1.24	0.62	1.27	0.71
cm42a	1.06	1.52	0.55	1.15	0.90
cm85a	1.00	1.49	0.78	0.98	0.88
decod	1.00	1.16	0.62	0.92	0.77
des	1.01	0.99	0.99	0.99	0.99
f51m	1.10	0.79	0.93	0.87	0.80
frg2	1.00	0.96	0.99	1.00	0.95
i6	1.00	0.96	1.00	0.87	0.90
i7	1.00	0.96	0.99	1.00	0.93
k2	1.00	1.07	0.78	0.97	0.88
lal	1.00	1.04	0.97	0.99	0.97
pm1	1.00	1.17	0.84	1.00	0.92
term1	1.05	0.97	0.98	0.98	0.95
ttt2	0.99	1.03	0.90	0.98	0.95
vda	1.00	1.05	0.84	1.00	0.97
Avg	1.03	1.06	0.90	0.99	0.90