Logic Extraction and Factorization for Low Power*

Sasan Iman, Massoud Pedram

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

Abstract

This paper describes algebraic procedures for node extraction and factorization that target low power consumption. New power cost functions are introduced for the sum-of-products and factored form representations of functions. These cost functions are then used to guide the power optimization procedures. It is also shown that using the proposed SOP power cost function, all extractions resulting in a power reduction will not result in an increase in the number of literals in the network. The procedures described in this paper were implemented and results show 16% average improvement in power at the cost of 7% average increase in area.

1. Introduction

With the increasing use of portable computing devices, size and battery lifetime are becoming important factors in today's digital systems. Meanwhile the amount of data to be processed is increasing at a rapid pace. This also places a severe demand on the speed of digital devices. The circuit power is also becoming the limiting factor in the amount of logic that can be placed in a VLSI chip and the determining factor in the packaging cost. These considerations have resulted in a growing need for minimizing power in today's digital systems.

Power consumption in a digital circuit can be optimized at different stages of the design process. However in order to maximally reduce the circuit power consumption, power has to be considered at all levels of the design hierarchy. These include optimizations performed at the behavioral, logic and physical levels [2][11][9][1].

Logic optimization based on algebraic operations has provided an efficient method for minimizing the area of a Boolean network. Algebraic based extraction algorithms [5] have provided fast methods for identifying logic sharing among different nodes. Substitution algorithms have helped in reducing the area by taking advantage of functions that have already been implemented. Algebraic factorization algorithms have provided fast techniques for estimating the area of a network by computing the number of literals in the factored form of the network. The application of these procedures during technology independent phase of logic synthesis has proved to be quite effective [7][12].

In this paper we address the problem of reducing power consumption during the technology independent phase of the logic synthesis using algebraic techniques. We first define new cost functions that measure the power cost of Boolean functions and then using these cost functions we extend the existing algebraic techniques to minimize power consumption.

The rest of this paper is organized as follows. In section 2. we give background information for power optimization and propose new power cost functions. In section 3. we discuss kernel and cube extraction operations for area and power. Section 4. describes the basics of factorization and describes a procedure for performing power factorization. In section 5. we discuss the effect of algebraic substitution on the power cost of a Boolean network. Selective collapse targeting low power is presented in section 6. Results and concluding remarks are presented in sections 8. and 9.

* This research was supported in part by the Semiconductor Research Corp. under contract no. 94-DJ-559 and a grant from the Intel Corp.

2. Estimating power consumption of a node

In order to accurately estimate the power consumption of a function f, we need to estimate the switching activity and load at the output of the gate implementing f.

In this paper we use the signal probabilities of the primary inputs to compute the signal probability of each internal node by building its global OBDD [6]. 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, then $t(n)=2*p(n)*(1-p(n))$ where $t(n)$ is the switching activity of node n.

We also use the following definitions to compute the output loads for nodes in the network. Given a node n_i and fanin node n_j, we define the *factored load* of n_j with respect to n_i, $FL(n_j, n_i)$ 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 also define the *cube load* of n_j with respect to n_i, $CL(n_j,n_i)$ as the number of times variable n_j is used (in positive or negative form) in the sum-of-products form of node n_i.

Using these definitions we define two power cost functions for a node. Note that the cost functions presented here represent the contribution of a node to the power consumption of the network since they include both the power at the input and the output of the node.

The sum-of-products cost function gives the power consumed by the node if the node was to be implemented in a sum-of-products form. Given a node n_i with output function f and cubes $(c_1,.. c_N)$, the power cost of n_i in the sum of products form is computed as:

$$P^{SOP}_{n_i} = t(f) \cdot \sum_{n_k \in fanouts(n_i)} CL(n_i, n_k) + \sum_{c_m \in cubes(n_i)} t(c_m) + \sum_{n_j \in fanins(n_i)} t(n_j) \cdot CL(n_j, n_i) \quad (1)$$

The first term in this equation corresponds to the power consumption at the output of the node. The second term corresponds to the power consumption at the output of the gates implementing each product term of the function and the last term accounts for the power on the inputs to the node.

Given a node n_i, the power cost of the node in the factored form is computed as [3]:

$$P^{FAC}_{n_i} = t(f) \cdot \sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) + \sum_{n_m \in Internal(n_i)} t(n_m) + \sum_{n_j \in fanins(n_i)} t(n_j) \cdot FL(n_j, n_i) \quad (2)$$

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

Note that is equations (1) and (2), the load for internal nodes is assumed to be 1. This is a valid assumption since for SOP representation, each node implementing a product term only fans out to one node. The procedure for factorizing a node [12] and its extension to consider power also guarantee that each internal node has only one fanout node.

These cost functions are used at different stages of technology independent power optimization. The *SOP* power cost function is used to guide the algebraic extraction and decomposition algorithms since the underlying assumption for algebraic operations is a SOP form implementation. The FAC power cost is used to guide the procedure for selective collapse when the goal is to minimize the power of the network in the factored form.

3. Common sub-function extraction

Algebraic extraction identifies common kernel and cubes and introduces them as new nodes into a network in order to optimize the area cost of a network. These techniques use the SOP cost of nodes to minimize the total number of literals in the SOP form of the network. Algebraic extraction techniques utilizing kernels and cubes are used extensively and result in significant reduction in the area of the mapped network [5].

3.1. Motivation

In area optimization the area value of a kernel is defined as the reduction in the number of SOP literals in the network when the kernel is extracted. We define the power value of a kernel as the reduction in the network power assuming a SOP implementation.

Example 1:

Assume $F = a\ b\ c\ d\ e + d\ e\ f\ g\ h\ i + f\ g\ i\ j\ k$. Also assume $p(f)=0.1$ and signal probability for all other fanins is 0.5.

All kernels of function F are:

$K_1 = a\ b\ c + f\ g\ h\ i$	*co-kernel: d*
$K_2 = d\ e\ h + j\ k$	*co-kernel: e f h*

Function F can be represented as:

$F_1 = K_1\ d\ e + f\ g\ i\ j\ k$

or $F_2 = a\ b\ c\ d\ e + K_2\ f\ g\ i$

Assuming an SOP implementation, extracting K_1 will reduce the number of literals by 1 and the power by 0.523. Extracting K_2 will reduce the number of literals by 2 and the power by 0.137.

■

For area optimization, maximum reduction in literals is obtained by extracting a kernel with maximum value. Extracting a kernel with maximum area value does not however always result in maximal reduction in the SOP form power of the network as illustrated in the previous example. In this example extracting K_2 will result in maximal reduction in the number of literals in the network. However more power reduction is obtained if K_1 is extracted.

3.2. Previous work

In [8] a modification of kernel extraction algorithm is presented which generates multiple level circuits with low power consumption. The procedure is as follows.

Let $d=d(v_1,...v_M)$, $M>0$ be a common sub-expression of function $F=(f_1,...f_L)$, $L>1$. Let $(J_1,...J_P)$, $P>0$ be the nodes internal to d. When d is factored out of f_i, the signal probabilities and switching activities at all nodes of the network remain unchanged. However the load at the output of the driver gates $(v_1,..v_M)$ change. Each gate now drives $L-1$ fewer gates. At the same time, since there is only one copy of d instead of L copies, there are $L-1$ fewer copies of internal nodes $(J_1,..J_P)$. The power saving in extracting d is thus given as:

$$(L-1)\cdot\left(\sum_{(i=1)}^{M} t(v_i)\cdot n_{v_i} + \sum_{(i=1)}^{P} t(J_i)\cdot n_{J_i}\right) \qquad (3)$$

where n_{vi} gives the number of gates belonging to d and driven by signal v_i and n_{Ji} is the number of gates internal to d and driven by signal J_i. One shortcoming of this method is that first a factored form for the functions is assumed and the sub-expressions are extracted based on these factored form representations. However once these sub-expressions are extracted, they will not necessarily have the assumed factorized form. This introduces an inconsistency in the flow of the procedure which will potentially lead to inconsistent results.

3.3. Kernel extraction targeting low Power

In the following, we describe an alternative approach for computing the power value for extracting a common sub-expression from a set of boolean equations. This power value uses the power cost of a node in the SOP form to compute the value for extraction. Using power cost in SOP form is consistent with the assumption made for computing the literal-savings value of a candidate divisor during algebraic extraction.

Consider a multiple-output Boolean function $F=\{f_1,..,f_L\}$ with cubes $(c_1,...,c_N)$ and input set $(v_1,...v_M)$. Let $D=d_1+..+d_P$ represent a kernel of function F used as a divisor. Also assume $Q=\{q_1,..q_R\}$ is the set of co-kernels for kernel D in function F. The area value for extracting D is given by equation (4). In this equation the first term accounts for literals saved by not repeating the kernel, the second term accounts for literals saved by not repeating the co-kernels and the last term accounts for the number of literals introduced by extracting kernel *D [7]*.

$$\left((R-1)\cdot\sum_{(i=1)}^{P} lit(d_i)\right)+\left((P-1)\cdot\sum_{(i=1)}^{R} lit(q_i)\right)-R \qquad (4)$$

We compute the power_value for extracting kernel D using equation (5). In this equation, the first term accounts for the reduction in the load on the inputs of the kernel. The second term accounts for the reduction in the load on the inputs to the co-kernels of the given kernel. The third term accounts for the cubes of the function that are removed from the original SOP representation of the function. (Note that in this representation $(d_i.q_j)$ corresponds to a cube of the original SOP representation of the function.) The fourth term corresponds to power consumption at the output of the new node which is inserted into the network. The fifth term corresponds to the power consumption at the output of the cubes of the new node that is inserted in the network. The last term accounts for the power at the output of the new cubes inserted in the SOP representation of function F.

$$\begin{aligned}&\left((R-1)\cdot\sum_{(i=1)}^{M} t(v_i)\cdot CL(v_i,D)\right)\\&+\left((P-1)\cdot\sum_{(i=1)}^{R}\sum_{(j=1)}^{M} t(v_j)\cdot CL(v_j,q_i)\right)\\&+\left(\sum_{(i=1)}^{P}\sum_{(j=1)}^{R} t(d_iq_j)\right)-(R\cdot t(D))\\&-\left(\sum_{(i=1)}^{P} t(d_i)\right)-\left(\sum_{(j=1)}^{R} t(Dq_j)\right)\end{aligned} \qquad (5)$$

The following example uses equations (4) and (5) to compute the area and power values for extracting a kernel.

Example 2:

Assume:

$F_1 = a\,x\,y + a\,u\,w + v\,z$

$F_2 = b\,c\,x\,y + b\,c\,u\,w = v\,z$

and $D = x\,y + u\,w \Rightarrow q_1 = a, q_2 = bc$ and $R = P = 2$

Also assume the following signal probabilities for the circuit inputs:

$p(a) = 0.97$	$p(u) = 0.91$	$p(x) = 0.67$
$p(b) = 0.02$	$p(v) = 0.93$	$p(y) = 0.47$
$p(c) = 0.51$	$p(w) = 0.35$	$p(z) = 0.65$

Using the given values, we compute the switching activity for the following product terms and functions:

$t(axy) = 0.424$	$t(auw) = 0.427$	$t(vz) = 0.478$
$t(bcxy) = 0.006$	$t(bcuw) = 0.006$	
$t(aD) = 0.500$	$t(bcD) = 0.011$	
$t(xy) = 0.431$	$t(uw) = 0.439$	
$t(D) = 0.498$	$t(F_1) = 0.309$	$t(F_2)= 0.011$

If D is extracted from F_1 and F_2, then

$F_1 = a\,D + v\,z, \qquad F_2 = b\,c\,D$

$D = x\,y + u\,w + v\,z$

Using equation (4), the area value for kernel D is computed as:

$(2\text{-}1)*4 + (2\text{-}1)*(1+2) - 2 = 5$

In fact the number of literals of the functions is reduced by 5 after D is extracted.

Using equation (5), the power value for this extraction is computed as follows:

$(2\text{-}1) * (t(x) + t(y) + t(u) + t(w))$
$+ (2\text{-}1) * (t(a) + t(b) + t(c))$
$+ (t(axy) + t(auw) + t(bcxy) + t(bcuw))$
$- 2 * t(D)$
$- (t(xy) + t(uw))$
$- (t(aD) + t(bcD))$

where each line in this equation corresponds to one term in equation (5). Then

power_value $= 1.559+0.597+0.863-0.996 -0.865-0.511$
$= 0.647$

■

Low power kernel extraction proceeds as follows: We first compute K, the set of all kernels for all the nodes in the network using a procedure described in [7]. We then generate D, the set of all kernel intersections for kernels in K. We then extract the sub-expression $D_i \in D$ which has maximum power value. This operation is performed while there exists a sub-expression D_i with a positive power value.

The kernel extraction procedure as described here minimizes the power by reducing load on high activity nodes and also introducing new nodes which have lower output activities. These conditions guarantee a reduction in power if the final area of the power optimized circuit is not significantly larger than the area of the circuit optimized for area. The following lemma shows that for all extractions that improve the power cost of the network as computed in equation (5), the literal count of the network will not be degraded.

Lemma.1. Given a set of functions $F=\{f_1, .., f_L\}$, and a candidate sub-expression D with co-kernels $Q=\{q_1,..q_1\}$, if power value of D is greater than or equal to zero then area value of D is greater than or equal to zero.

Proof: Assume K is the number of literals in kernel D and C is the number of literals in co-kernels of D. Then area value of D as given in equation (4) can be negative only when $K=1$ or when $R=2$ and $P=1$. Equation (5) shows that under these conditions power value of D will also be negative. This means that a negative area value results in a negative power value which in turn implies the claim of the lemma.

■

3.4. Cube extraction

Consider a Boolean function $F=\{f_1,..,f_L\}$ with cubes $(c_1,...,c_N)$ and input set $(v_1,...v_M)$. Let $D = c_i \cap c_j$ be a sub-cube of the function with T literals that is shared by $R(>1)$ cubes of the function. The area value for extracting D is given by [7]:

$$(R-1)\cdot(T-1)-1 \tag{6}$$

We compute the power value for extracting cube D using equation (7). The first term in this equation accounts for the load reduction on the inputs fanning out to D. The second term accounts for the power introduced in the network by adding a gate into the network. The load for this new gate is $CL(f,D)$ which is equal to R.

$$\left((R-1)\cdot\sum_{(i=1)}^{M} t(v_i)\cdot CL(v_i, D)\right) - R\cdot t(D) \tag{7}$$

The procedure for power cube extraction proceeds as follows: We first compute C, the set of all cube intersections for all the nodes in the network using a procedure described in [7]. We then extract the sub-expression $C_i \in C$ which has maximum power value. This operation is performed while there exists a sub-expression C_i with a positive power value.

As with kernel extraction, the following lemma guarantees that any cube extraction for power will not degrade the number of literals in the network.

Lemma.2. Given a set of functions $F=\{f_1, .., f_L\}$, and cube C with T literals to be extracted from F, if the power value of C is greater than or equal to zero then area value of C is greater than or equal to zero.

Proof: It is similar to the proof for Lemma 1.

■

3.5. Quick power extract

The shortcoming of the power value as presented in equations (4) and (5) is that for each sub-expression being extracted, the switching activity of all the cubes being removed and being inserted have to be computed. Correct computation of the switching activity for any internal node or function (under a zero delay model) [9] [4] requires that the global OBDD and then the signal probability of the function be computed. This operation proves to be very time consuming. In order to speed up the extraction procedure we use the signal probability values on immediate fanins of the node to approximately compute their switching activity.

4. Factorizing logic functions

Factorization is the process of deriving a factored form from a sum-of-products form of a function. For example if $F=a.b+a.c+b.c$ then one possible factorization of F is $a.(b+c)+b.c$. Since the area of a Boolean network is more accurately estimated by the number of literals in the factored form of the network, efficient factoring algorithms are needed to guide the optimization problems.

In [12] a recursive procedure called *generic_factor,* is presented for factorizing a function. At each step of the recursion the function F passed to the procedure is transformed into $F=Q.D+R$ where D is the divisor, Q is the quotient and R is the remainder of dividing F by D. The procedure guarantees that the resulting factorized form of F is maximally factorized.

The procedure *DIVISOR* passed to the function is used to find a candidate divisor for the function. By changing this procedure, trade-offs in terms of speed and quality of results can be made. Quotient Q is computed by performing weak division[12] on F and D.

Two commonly used factorization techniques are "*quick*" and "*good*" factorization which are performed by using different *DIVISOR* procedures with the *generic_factor* routine. In "*good*" factorization, at each level of the recursion in *generic_factor*, a kernel is returned by *DIVISOR* which results in maximum reduction in the number of literals in the SOP form of the function being factored. This kernel is selected by generating the set of all kernels and then selecting the one with maximum area value.

In "*quick*" factorization, *DIVISOR* returns a level 0 kernel [7] of the function which is generated using a recursive procedure called *best_literal*. At each step of the recursion, *best_literal* divides the function by the most occurring literal l in the SOP form. This choice of l for the given function F guarantees that at each step of the recursion, the number of literals in the resulting expression, $Q.l + R$ is maximally reduced.

4.1. Factorizing for power

The goal of factorization for low power is to produce a factored from where the weighted sum of the literals in the factored form is minimized where each literal is weighted by its switching activity. In this paper, the procedure *generic_factor* is used to ensure that the resulting factored forms are maximally factored. The procedures for finding a divisor however, are modified to take into account the switching activity of the literals in the factored form representation.

In "*good_power*" factorization *DIVISOR* finds the best power divisor by generating the set of all kernels in the function and then returning the kernel with the best power value. The power value of kernels are computed using equation (5).

In "*quick_power*" factorization, DIVISOR will return a level 0 kernel as is done in "*quick*" factorization. The procedure for generating this level 0 kernel however is modified to take into account the switching activity of the literals. procedure *best_power_literal* is a variation of *best_literal* where at each step of the recursion the function F is divided by the literal l with the highest value $CL(l,F)*t(l)$.

*Lemma.3. Given an algebraic expression F, selecting literal l which maximizes CL(l,F)*t(l) will result in minimum weighted sum of the literals in the factored form of the expression Q.l+R obtained by dividing l into F.*

Proof: Reduction in the weighted sum of the literals is given by $(CL(l,F)-1)*t(l)$. Maximally reducing the weighted sum of literals is thus equivalent to maximizing $CL(l,F)*t(l)$.

■

5. Substitution

Substitution of a function G into F is the process of re-expressing F in terms of variable G and the original inputs of the function F[12]. Substitution can be performed using algebraic or Boolean division algorithms. Even though Boolean division in general yields better results, algebraic division is a much faster heuristic which produces comparable results. If algebraic division is used for substituting variable G into function F and the quotient of the result is not 0, then we can conclude that G was a sub-expression of function F. In fact if a non-trivial expression G (a function other than buffers or wires) can be substituted in function F using an algebraic division procedure (i.e. weak division), then the area value as defined in equation (4) is always greater than zero. The important consequence of this is that if a function G can be substituted into a function F, then it is guaranteed that this substitution will result in a decrease in the number of literals of the network in the SOP form.

Substitution is also used to minimize the network power consumption. However substitution does not always guarantee a reduction in the power cost of the network. This means that an operation which decreases the number of literals in the network can potentially increase the power cost of the network in the SOP form.

Example 3:

Assume $F_1 = a\ b\ c$ and $F_2 = a\ b$

also $p(a) = p(b) = p(c) = 0.9, p(F_2) = 0.5$

Note that $p(F_2) \neq p(a)*p(b)$. The cause for this is spatial dependence between inputs a and b[4].

F_1 can be expressed in terms of F_2 as follows:

$$F_1 = F_2\ c$$

After the substitution, only the input plane for F_1 is changed in the network and the power at the input of F_1 is increased from 0.54 to 0.68 even though literal count is decreased.

■

Once it has been determined that a node n_j can be substituted into a node n_i, we compute the power cost function of n_i as given in equation (1) for both SOP implementations of n_i before and after substituting n_j. We proceed with the substitution if the power cost of n_i is reduced after substitution.

6. Selective collapse

Selective collapse is the process of selectively eliminating nodes in a network in order to reduce the area cost of the network. Selective collapse is performed on an initial Boolean network to provide a better starting point for the extraction procedures. This is done since the initial network might have factors identified which are not good candidates for extraction. During area optimization, two value functions are used to decide if a node should be eliminated by collapsing it into its fanout nodes. One is the sum-of-products value of the node. This value is the reduction in the number of literals in the sum-of-products form of the network if the node is collapsed into all its fanout nodes. The area value and power value for a node n_i are given by equations (4) and (5).

The second value function is the factored form value. This value gives the reduction in the number of literals in the factored form of the network if the node is collapsed into its fanout nodes. The area value in the factored form of a node is defined as follows[7]:

$$\left(L(n_i) \cdot \sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) \right) - \left(\sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) \right) - L(n_i) \qquad (8)$$

where $L(n_i)$ is the number of literals in the factored form of the node n_i. The first term in this equation accounts for the duplication of the literals in the factored expression of n_i once it is collapsed into its fanouts. The second term is the decrease in the number of literals by removing the literals corresponding to n_i from its fanout nodes. The last term account for the removal of n_i from the network where $L(n_i)$ literals are removed from the network.

We define the factored form power value for selective elimination as follows:

$$\left(\sum_{n_j \in fanins(n_i)} t(n_j) \cdot FL(n_j, n_i) \right) \cdot \left(\sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) - 1 \right) - \left(t(n_i) \cdot \sum_{n_k \in fanouts(n_i)} FL(n_i, n_k) \right) \quad (9)$$

The first term in this equation gives an estimate of the power added to the network by duplicating the node function. The second term accounts for the power at the output of the node which is removed from the network.

7. Optimization scripts

Logic optimization script *script.algebraic* provided in the SIS package is used as the starting point for a power optimization script. Algebraic commands in this script are replaced by equivalent commands for power to generate *script.power_alg*.

A power recovery stage is also performed at the end of *script.power_alg*. The following example illustrates the motivation behind this step.

Example 4:

Assume $F = a\,b\,c + a\,d\,e + e\,f\,g$. Also assume signal probability for all inputs is 0.5.

Function F has two kernels $K_1 = b\,c + d\,e$ and $K_2 = a\,d + f\,g$.

The *power value*(K_1) = *power_value*$(K_2) = -0.646$. This means that any further decomposition of this function will result in an increase in the power consumption of the network.

■

In *script.algebraic* all nodes are fully decomposed before mapping is started. However this will not be true for *script.power_alg* since some decompositions will result in an increase in the power. A power recovery stage is added after the decomposition procedure in *script.power_alg* where all nodes are first decomposed into n-input NAND gates. Elimination for power is then performed to eliminate all nodes which increase the node node in the factored form. This operation is followed by extraction and decomposition for power.

8. Results

The procedures outlined in this paper have been implemented in SIS and *script.power_alg* has been created.

Each circuit in the benchmark set was first optimized for area using *script.algebraic* and then mapped for minimum power using an industrial library and the power driven technology mapper presented in [10]. The same circuits were then optimized for minimum power using *script.power_alg* and then mapped using the same library and technology mapper. . It is difficult to accurately estimate the switching activity under a real delay model for circuits before technology mapping. Therefore we adopt a zero delay model for computing the node switching activities. Table 1 shows the results when the circuits are optimized using *script.algebraic*. Column 1, 2 give the power and area of the circuit after mapping. Column 3 gives the average switching activity of internal nodes in the network after technology mapping. Power is measured using the library loads and area is measured as the sum of the gate areas. Columns 4, 5 and 6 show the power, area and average switching activity for the nodes in the network before the technology mapping is started. Here power is measured assuming an SOP implementation and area is given as the number of literals in the network. The results in Table 2 are also normalized with respect to the results in Table 1. As the results show, power after mapping has been reduced by 21% at a cost of increasing the area by 7%. The average switching activity has also been reduced by 24%. As shown by the results, the extraction procedure has been very effective in introducing new nodes in the network with small switching activities. There is also a 22% decrease in the total sum of the power for all circuits being optimized at the expense of increasing the total area by 10%.

Table 3 and 4 show the same results starting with multi-level examples. Results show an average 10% improvement in power at the cost of increasing the area by 5%. Average switching activity in the network is also reduced by 12%. The total sum of power over these circuits is also reduced by 6% at the expense of increasing the area by 6%.

The run-time for *script.power_alg* is much more than that of *script.algebraic*. This is due to the fact that at start up and after each extract(but not during candidate kernel evaluation) global OBDDs were built for computing the switching activities in the network. More efficient techniques have been proposed for computing the network switching activities [4]. Using these efficient methods, the run time of *script.power_alg* can be significantly improve.

9. Conclusions

In this paper we have proposed a unified approach to power optimization using algebraic based methods. The results show that in general it is possible to slightly increase the network area in order to reduce the power consumption of the technology mapped network. This is accomplished by reducing load on high activity nets and also by introducing new nodes which have a lower switching activity. Power cost functions were proposed to find the quality of extractions performed for low power which proved to be quite effective.

10. References

[1] A. P. Chandrakasan, M. Potkonjak, , J. Rabaey, R.W. Broderson, HYPER_LP: A System for power minimization using architectural transformations, ICCAD, Nov. 1992, pages 300-303.

[2] 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.

[3] S. Iman, M. Pedram, Multi-level network optimization for low power. ICCAD'94, November 1994, pages 372-377.

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

[5] R. Brayton, C. McMullen. The decomposition and factorization of boolean expressions. In Proc. Int. Symp. Circ. Sys. (ISCAS-82)pages 49-54, 1982.

[6] F.N. Najm, R. Burch, P. Yang, I. Hajj, Probabilistic simulation for reliability analysis of CMOS VLSI circuits. IEEE transactions on CAD, 1990, volume 9, pages 439-450.

[7] R. Rudell, Logic Synthesis for VLSI Design, Ph.D. Thesis, UC, Berkeley, 1989.

[8] K. Roy, S. C.Prasad. Circuit activity based logic synthesis for low power reliable operations. IEEE Transactions on VLSI systems. 1(4):503-513, December 1993.

[9] A. A. Shen, A. Ghosh, S. Devadas, K. Keutzer, On average power dissipation and random pattern testability of CMOS Combinational Logic Networks. ICCAD, November 1992, pages 402-407.

[10] C-Y. Tsui, M. Pedram, A. M. Despain, Technology decomposition and mapping targeting low power dissipation. DAC'93, pages 68-73, 1993.

[11] H. Vaishnav, M. Pedram, PCUBE: A performance driven placement algorithm for low power designs, EuroDac, Sept. 1993, pages 72-77.

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

Table 1.script.algebraic for two-level circuits

	Post-Map			Pre-Map		
ex	1	2	3	4	5	6
apex2	0.88	48.6	0.13	108.	379	0.22
apex4	4.39	414.	0.06	496.	272	0.11
b12	0.19	11.0	0.14	22.5	93	0.21
clip	0.41	17.1	0.24	50.1	141	0.30
cps	1.80	174.	0.08	207.	118	0.14
duke2	0.86	60.3	0.12	87.8	418	0.15
ex4	1.38	67.3	0.16	176.	556	0.21
inc	0.46	18.8	0.18	54.6	148	0.30
misex2	0.23	15.7	0.10	23.2	109	0.10
misex3c	0.97	85.5	0.08	111.	601	0.15
misex3	1.88	118.	0.11	220.	844	0.20
pdc	1.47	81.1	0.15	150.	570	0.17
rd53	0.28	7.00	0.38	33.0	62	0.46
rd73	0.21	19.1	0.10	29.8	143	0.21
rd84	0.43	24.0	0.15	53.4	178	0.27
sao2	0.47	21.7	0.16	60.6	182	0.25
spla	1.39	86.9	0.13	154.	616	0.17
squar5	0.11	8.96	0.12	13.0	70	0.19

Table 2.script.power_alg for two-level circuits

	Post-Map			Pre-Map		
ex	1	2	3	4	5	6
apex2	0.75	1.18	0.65	0.68	1.08	0.50
apex4	0.74	1.06	0.81	0.72	1.09	0.62
b12	0.96	1.14	0.68	0.92	1.10	0.74
clip	0.79	0.90	0.81	0.82	0.94	0.80
cps	0.85	1.16	0.76	0.79	1.14	0.61
duke2	0.76	1.11	0.75	0.71	1.06	0.52
ex4	0.94	1.07	0.84	0.91	1.05	0.82
inc	0.70	0.92	0.87	0.66	0.95	0.72
misex2	0.94	1.01	0.94	0.95	1.02	0.84
misex3	0.77	1.14	0.65	0.75	1.15	0.55
misex3c	0.78	1.07	0.70	0.82	1.10	0.63
pdc	0.80	1.08	0.67	0.90	1.10	0.73
rd53	0.65	0.81	0.89	0.59	0.69	0.79
rd73	0.78	1.08	0.65	0.72	1.15	0.53
rd84	0.76	1.19	0.63	0.78	1.28	0.56
sao2	0.67	1.19	0.52	0.62	1.03	0.44
spla	0.73	1.16	0.61	0.73	1.11	0.55
squar5	0.80	0.95	0.84	0.84	1.00	0.71
Avg	**0.79**	**1.07**	**0.74**	**0.77**	**1.06**	**0.65**

Table 3.script.algebraic for multi-level circuits

	Post-Map			Pre-Map		
ex	1	2	3	4	5	6
C1355	1.80	69.6	0.27	214.1	556	0.290
C1908	1.50	84.4	0.19	170.2	565	0.238
C432	0.62	36.5	0.14	75.06	250	0.178
C880	1.23	57.3	0.19	152.7	445	0.256
alu2	0.88	64.4	0.10	105.0	467	0.208
alu4	1.28	130.	0.08	102.9	864	0.108
apex6	2.19	113.2	0.16	272.2	827	0.300
b9	0.32	17.4	0.16	38.66	131	0.251
dalu	2.43	158.	0.14	288.1	1239	0.193
des	8.65	568.	0.12	1068.	3765	0.280
f51m	0.35	18.4	0.17	43.28	140	0.298
frg1	0.40	17.8	0.20	46.73	146	0.277
k2	1.39	174.	0.05	119.8	1069	0.058
rot	1.62	99.8	0.15	204.9	773	0.235
t481	0.85	122.	0.06	70.93	915	0.052
ttt2	0.47	26.11	0.13	57.41	214	0.188
x2	0.15	6.93	0.18	14.98	54	0.290
9symml	0.61	34.1	0.12	79.68	266	0.262

Table 4.script.power_alg for Multi-level circuits

	Post-Map			Pre-Map		
ex	1	2	3	4	5	6
C1355	0.91	0.98	0.87	0.97	1.00	0.95
C1908	0.94	0.90	1.00	0.94	1.01	0.89
C432	0.92	0.90	1.10	0.96	1.01	1.03
C880	0.92	1.00	0.94	0.91	0.99	0.88
alu2	0.85	1.01	0.78	0.87	1.05	0.78
alu4	0.65	1.07	0.57	0.87	1.10	0.68
apex6	0.98	1.05	0.95	0.93	1.06	0.89
b9	0.99	0.99	1.03	0.99	1.08	0.90
dalu	1.07	1.14	0.90	1.12	1.12	0.91
des	0.99	1.02	1.12	0.94	1.05	0.88
f51m	0.89	1.11	0.74	0.96	1.23	0.72
frg1	0.85	1.05	0.72	0.87	1.10	0.72
k2	0.94	1.24	0.89	0.71	1.28	0.57
rot	0.99	1.00	1.02	0.93	1.01	0.91
t481	0.78	1.07	0.68	0.98	0.99	0.82
ttt2	0.85	1.15	0.78	0.86	1.11	0.79
x2	0.89	0.97	1.01	0.88	0.94	0.89
9symml	0.83	1.16	0.75	0.77	1.14	0.61
Avg	0.90	1.05	0.88	0.91	1.07	0.82