Re-mapping for low power under tight timing constraints

P. Vuillod L. Benini G. De Micheli
CSL Stanford University, USA

Abstract

In this paper[1] we propose a novel approach to synthesis for low power under tight timing constraints. Starting from a mapped netlist, we apply a powerful *generalized matching* algorithm based on Boolean relations that allows us to find reduced-power replacements for clusters of more than one cell. Our approach is robust and scales well with circuit size: it has been tested on all largest examples of the MCNC91 benchmark suite. In average, power is reduced by more than 17% with no speed penalty compared to minimum delay implementations. Area is virtually unchanged.

1 Introduction

The problem of reducing the power consumption of combinational CMOS logic has received considerable attention in the last few years [1]. The most comprehensive approach in the literature is probably the work presented in [2, 3, 4] which was integrated in a toolset for power optimization called POSE [5].

The key idea in POSE is that classic logic optimization techniques (e.g. factorization, decomposition and others) can be modified to take into account power as a cost function. Unfortunately, in some cases, transformations that minimize power affect adversely area and timing. The approach taken by POSE is to trade off some controlled amount of area/speed for a consistent power improvement. In many cases, such trade-off is not necessary because the power-optimal design is a good solution even for area and timing. However, there is no guarantee that this is *always* the case.

Speed is the primary concern in state-of-the art digital systems. The timing budget for a combinational logic block is specified at the architectural level and becomes a constraint for synthesis. In the majority of the practical cases the designer (and the synthesis tool) struggles to satisfy the timing constraints. Once timing constraints are met, secondary cost functions can be optimized. Power is one of such cost functions. Since satisfying timing constraints is a primary objective, it is not possible to trade off performance for power, no matter how big the power savings may be.

We call this problem *power minimization under tight timing constraints.* Such constrained optimization problem is often encountered in the design practice. In this paper we propose a solution to power minimization under timing constraints, where we minimize power without trading off performance. Moreover, we focus on the last stages of the synthesis process, namely we target the power optimization of a mapped netlist. Post-mapping optimization is often called *re-mapping.*

Several post-mapping power reduction techniques have been proposed in the literature [6, 7, 8]. Our approach differs from previous ones in: type of transformations, re-mapping engine for generating candidate transformation and starting point for optimization.

Our power optimizer relies on a robust and accurate power estimation engine based on *bit-parallel simulation* [10] (BPS) and probabilistic estimation. More specifically, we rely on bit-parallel simulation for power estimation of the entire network, while we base our local estimates on fast probabilistic estimation. Depending on the efficiency requirements and the degree of confidence on the quality of the results of the tool, we can specify different estimation modes that test the effectiveness of our transformations with varying degrees of accuracy.

We tested our re-mapping tool on a large set of examples, including the largest benchmarks in the MCNC91 [19] suite for which, to our knowledge, no power optimization results have been presented in the literature. In average, power was reduced by more than 17% with no performance penalty, starting from tightly timing-constrained mapped networks. Power reductions are *not* due to decreases in area since we perform area recovery on the timing-constrained networks before applying re-mapping. Indeed, area is almost unchanged (average 0.26% decrease). Run times are in the order of those required for technology-independent optimizations and library binding.

2 Background

We assume that the reader is familiar with Boolean algebra and BDD based Boolean manipulation (see [13] for a review). We denote vectors and matrices in bold, i.e., $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$.

We use the symbols $\forall_x f = f_x \cdot f'_x$ and $\exists_x f = f_x + f'_x$ to designate respectively the *consensus* and the *smoothing* of Boolean function f with respect to variable x.

We consider Boolean functions that model a portion (or cluster) of the circuit. They are called *cluster functions.* We denote by $\mathbf{f} = [f_1, f_2, ..., f_n]^T$ a generic multi-output cluster function. We call *pattern function* a combinational function modeling a library cell, and we use g to represent a generic single-output pattern function.

In [11] we introduced the concept of *generalized matching* (GM). Generalized matching extends the Boolean relation-based approach [9, 16] to the technology dependent part of the synthesis flow. GM has two key advantages with respect to traditional single-output Boolean matching techniques, namely (i) expressing the

[1]L. Benini was supported by NSF under contract MIP-942119. P. Vuillod was on leave from INPG - CSI, FRANCE.

degrees of freedom for matching with a Boolean relation which is more powerful than *don't cares* [9]; (ii) concurrently matching multiple single-output cells. As a result, GM finds matchings that *cannot be found* with any traditional Boolean matching technique.

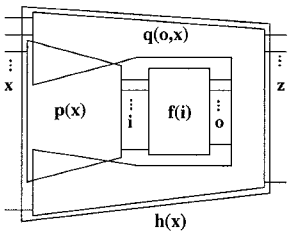

Figure 1: A multi-output cluster (**f**) function and its neighborhood **h**.

Let us consider a multi-output cluster function **f** embedded in a logic network as shown in Figure 1. We adopt a formalism similar to that used by Watanabe and co-authors [16]. We can compute a Boolean relation representing the complete set of *compatible functions* of **f**, i.e. functions that can implement **f** without changing the input-output behavior of **h**. Watanabe et al. showed that the characteristic function $\mathcal{F}$ of the Boolean relation can be obtained with the following formula [16]:

$$\mathcal{F}(\mathbf{i},\mathbf{o}) = \forall_{\mathbf{x},\mathbf{z}}[(P(\mathbf{x},\mathbf{i}) \cdot Q(\mathbf{o},\mathbf{x},\mathbf{z})) \Rightarrow H(\mathbf{z},\mathbf{x})] \quad (1)$$

where P, Q and H are the characteristic functions of **p**, **q** and **h**.

The GM problem consists in finding all possible sets of n library cells that can implement one of the functions represented by $\mathcal{F}$ [11]. To accomplish this task we define the concept of *quotient function* $L(\mathbf{i},\mathbf{c})$ for our technology library. The pictorial representation of the quotient function is shown in Figure 2 for a simple library with $N_{lib} = 3$ cells, g_1, g_2 and g_3.

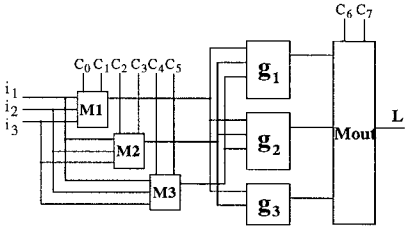

Figure 2: Quotient function of the target library.

In the figure, the blocks $M1$, $M2$, $M3$ and $Mout$ represent multiplexers, with control inputs $\mathbf{c} = [c_0, c_1, ..., c_7]^T$. The first three multiplexers control the input pin assignments. By changing the control inputs we can control how the external inputs are connected to the pins of the cells. Multiplexer $Mout$ controls cell selection: it selects which cell is connected to the output.

In order to perform generalized matching, we need to check if a n-output cluster function $\mathbf{f}(\mathbf{i})$ can be replaced by n library cells. For the sake of simplicity, we restrict to $n = 2$. We can express GM with a Boolean formula in L and $\mathcal{F}$ [11]:

$$M(\mathbf{c}) = \forall_{\mathbf{i}} \exists_{\mathbf{o}} \left(\mathcal{F}(\mathbf{i},\mathbf{o})(L(\mathbf{i},\mathbf{c}_1) \overline{\oplus}\ o_1)(L(\mathbf{i},\mathbf{c}_2) \overline{\oplus}\ o_2)\right) \quad (2)$$

Where $\mathcal{F}$ is the Boolean relation for the cluster, L is the quotient function. Notice that for each output, we have distinct sets of control variables, hence $\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2]$. This is because each output of **f** can be matched by a different cell with different input assignments. $M(\mathbf{c})$ is called *matching function*; its ON-set denotes all possible assignments of the cluster to two library cells with the property that the new implementation of the cluster function can replace the old one without changing the behavior observed at the output of **h**.

2.1 The re-mapping approach

Our power optimization strategy is based on a *re-mapping* approach. Starting from a mapped circuit, we apply our optimization engine to specific regions of the mapped netlist where local improvements are more likely. Once a target region is selected, it is optimized and, if optimization is successful, one or more cells are replaced with lower cost alternatives. Moreover, local wiring can be changed: the new cells may have different inputs.

A good choice of the target regions in the mapped netlist is paramount for the success of the re-mapping strategy. We focus on *multiple fanout points* (MFPs). The enumeration of the MFPs is done by traversing the network in a backward *breadth-first* fashion starting from the output and moving toward the inputs. Whenever a MFP is found, a set of cluster functions is generated by taking (i) sets of gates in the fanout of the MFP, (ii) the gate in the fanin of the MFP and sets of gates in its fanout. When all MFPs in the network have been explored, the process is restarted and new passes through the network are made until no further improvement can be achieved. Although the algorithm can generate cluster functions with more than two outputs, we will consider here only two-output clusters for the sake of explanation. Once a cluster has been generated, its neighborhood (i.e., function **h**) is constructed by taking a subset of its transitive fanin and fanout up to a user-defined depth. The algorithms for cluster generation and neighborhood construction are described in a companion paper [15]. For the remainder of this work, we assume that a multi-output cluster function is selected, its neighborhood has been generated and Boolean relation $\mathcal{F}$ representing the degrees of freedom available for generalized matching has been extracted from the neighborhood (using Equation 1).

Given a library and a Boolean relation $\mathcal{F}$, GM can be solved in principle by applying Equation 2 and computing $M(\mathbf{c})$. There are two fundamental problems with this straightforward approach. First, computing Equation 2 by simply composing BDD operators is extremely inefficient. Memory blowup is very common during conjunction of the quotient functions with $\mathcal{F}$ and during quantification. Second, even if we succeed in computing $M(\mathbf{c})$ we have simply solved a *decision problem* (i.e. found all matching assignments). We actually need to solve an *optimization problem*, i.e. finding among the matching assignments the ones that minimize power and respect the constraints. We could solve the optimization problem by enumerating the ON-set of $M(\mathbf{c})$ and by evaluating constraint and cost function for each minterm. It is easy to see that this solution is not practical, since the ON-set of $M(\mathbf{c})$ can be very large.

To address both these problems we have implemented an optimized matching procedure, based on two important concepts, namely *symbolic representation of cost*

function and constraints and *compression of the quotient functions through bounding*. The complete algorithm for GM is quite involved and it is described elsewhere [15].

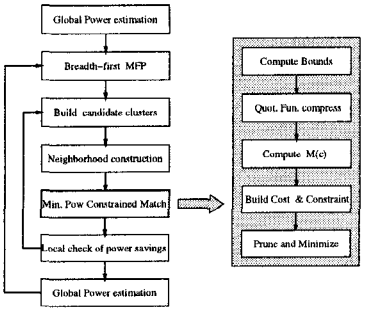

Figure 3: High-level flow of the re-mapping procedure

The high level flow diagram of the re-mapping procedure is shown in Figure 3. An initial global power estimation is performed (using BPS), then the breadth first iteration on MFPs is started. From each MFPs, candidate multi-output clusters are generated and the neighborhood is constructed (Boolean relation $\mathcal{F}$ is computed in this step). Matching starts by computing bounds to eliminate cells that are certainly sub-optimal, then the compressed quotient functions are built. The matching assignments are computed using Equation 2. The symbolic representation of timing constraint and power cost are built and the ON-set of $M(\mathbf{c})$ is first pruned by the constraint, then the minimum-cost minterms of the pruned M are selected. Replacement is performed if there is improvement with respect to the initial mapping. After each replacement, fast probabilistic simulation is applied to check the power savings. Finally, BPS-based power estimation is run every few replacements to check for global improvement. The breadth-first iteration is repeated until no further improvements are achieved.

3 Power minimization

This section presents the algorithmic details of GM for minimum power under delay constraints, i.e. the sequence of steps shown in the dark box of Figure 3. It can be skipped without compromising the high level understanding of our approach. The section is structured as follows. We first present a simplified symbolic power model. The second subsection shows how we bound the search space using the cost function. In the third subsection we address the computation of the timing constraints.

3.1 Symbolic Power Model

Remember that the ON-set of M in Equation 2 represents all possible matching assignments for the cluster function. We are interested in the ones that minimize power. In order to get the minimum power assignments in the ON-set of $M(\mathbf{c})$, we need to compute the power cost of each minterm. This task is efficiently carried out by ADD-based symbolic techniques. We consider in the following discussion only the power due to output switching activity (also known as *external power* or *switching power*). Internal power is not discussed for space reasons.

External power at a node o of a Boolean network is given by:

$$\mathcal{P}_{ext}(o) = K P_t(o) C(o) \frac{{V_{dd}}^2}{2} \tag{3}$$

Where $P_t(o)$ is the transition probability of node o, $C(o)$ the capacitance at node o, V_{dd} the supply voltage, and K a constant factor.

We consider a cluster with two outputs. The two outputs are bound by the Boolean relation $\mathcal{F}$. A target library function can be placed at one of the outputs o if it satisfies $\mathcal{F}(\mathbf{i}, \mathbf{o})$. The same holds for the other output. We want to analyze the gain in power of the replacement at one of the outputs o of the cluster. External power can change at the node o, because the transition probability $P_t(o)$ depends on the function at o. Observe that the variation is possible because, by exploiting the degrees of freedom expressed by $\mathcal{F}$, we may modify the function at o, and consequently the transition probability. The capacitance $C(o)$ is independent from the function at output o, because it depends only on the fanouts of o which are not modified by matching.

The computation of the transition probability is done symbolically as follows. Output node o is represented by the function $o = f(\mathbf{i})$. We call $\mathbf{i}$ and $\mathbf{i}^+$ two consecutive input patterns, their response at the output being $o = f(\mathbf{i})$ and $o^+ = f(\mathbf{i}^+)$. The transition probability of o, $P_t(o)$, is the probability that o switches, therefore it is $P(o \neq o^+)$. Consider now two consecutive input patterns $\mathbf{i}$ and $\mathbf{i}^+$. If the responses to these patterns o and o^+ are different, we observe a transition at the outputs. This event contributes to $P(o \neq o^+)$. It happens with a probability $P(\mathbf{i}, \mathbf{i}^+)$. The formula for the transition probability is the sum of all such events at the inputs. We obtain the following formula:

$$P(o \neq o^+) = \exists_{\mathbf{i},\mathbf{i}^+} (f(\mathbf{i}) \overline{\oplus} f(\mathbf{i}^+)) P(\mathbf{i}, \mathbf{i}^+) \tag{4}$$

If we assume spatial independence of the inputs and we neglect high order temporal correlations (i.e. correlations between patterns with more than one cycle time difference), the probability of two consecutive input vectors $P(\mathbf{i}, \mathbf{i}^+)$ is the product of the probability of each bit sequence of the vector.

The consequence of the transition probability change is the modification of power consumption at o, but at the fanout nodes of o as well. If the transition probability changes at o, its fanout nodes will have a new transition probability. However, we know from Section 2 that the output nodes $\mathbf{z}$ of the neighborhood have the same behavior regardless the matching. Therefore they have a constant transition probability and consequently, all the nodes in their fanout have a constant probability for this given matching. So the impact in the fanout nodes does not go further than $\mathbf{z}$. The impact of the change on the fanout nodes has to be taken in consideration, but it has only a limited scope in the circuit, namely, the nodes inside the neighborhood. We will discuss this issue in Section 4. For now, we assume that changes in transition activity of o do not sensibly affect the transition activity even for nodes within the neighborhood.

We analyzed the impact of a replacement at the outputs of the cluster. However, external power is modified at the inputs as well. Consider i, one of the inputs $\mathbf{i}$.

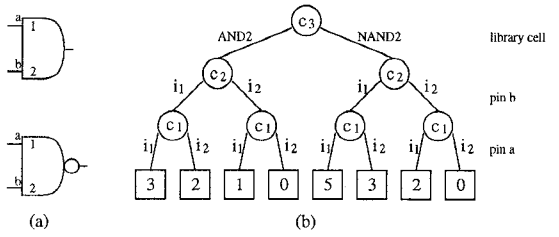

Figure 4: (a) Library cells (b) ADD of the capacitance at i_1 for any selection.

The power variation at the input is not due to the transition probability, because this latter may change only at the gate output. It is caused instead by the variations of the load $C(i)$ at the input when we change cell and pin assignment during re-mapping. The load at input i can be expressed as:

$$C(i) = C_i^{other} + C_i^1 + C_i^2 \tag{5}$$

Where C_i^1 (C_i^2) is the input capacitance of the cell whose output will be connected to o_1 (o_2) and to input i. (If i is not connected to the cell at o_1 (o_2), $C_i^1 = 0$ ($C_i^2 = 0$)). C_i^{other} is a constant, but C_i^1 and C_i^2 depend on the cell selection and the pin assignments.

To evaluate the power cost we need to compute the input capacitance and the output transition probability for each cell and pin assignment. Power cost is a function of the variables $\mathbf{c}$. For each minterm of the Boolean space of $\mathbf{c}$, we compute a power value. This can be easily and compactly handled by Algebraic Decision Diagrams [14] (ADDs). As we have shown before, we need to find the output transition probability for any cell configuration, and the capacitance at each input for any input assignment.

For a particular value of the control variables $\mathbf{c} = \mathbf{c}^*$, the restriction of the quotient function $L(\mathbf{i}, \mathbf{c}^*)$ represents a Boolean function of the inputs $\mathbf{i}$ alone: it is the function implemented by the library cell and the input pin assignment expressed by $\mathbf{c}^*$. For such function, we can compute P_t. Thus, we can extend the formula in Equation 3 and compute P_t for any possible value of $\mathbf{c}$ in a symbolic fashion using ADDs:

$$P_t(\mathbf{c}, o) = \exists_{\mathbf{i},\mathbf{i}^+}((L(\mathbf{i}, \mathbf{c})) \overline{\oplus}\ L(\mathbf{i}^+, \mathbf{c}))P(\mathbf{i}, \mathbf{i}^+) \tag{6}$$

With $P(\mathbf{i}, \mathbf{i}^+)$ being the ADD computing the probabilities of $(\mathbf{i}, \mathbf{i}^+)$, and L the BDD of the quotient function. $P_t(\mathbf{c}, o)$ represents how the transition probability of output o changes as a function of the control variables (that select different implementations for o).

The the function must take into account the effect of input capacitance. The ADD formulation has the same form as Equation 5, but C_i^1 and C_i^2 are ADDs function of $\mathbf{c}$. The ADD of the capacitance is then $C_i(\mathbf{c}) = C_i^1(\mathbf{c}) + C_i^2(\mathbf{c}) + C_i^{other}$. To compute $C_i(\mathbf{c})$, we need to build an ADD with the leaves containing the input capacitances of the pin selected by each ADD path.

Example 1 Consider cells AND2 and NAND2 Figure 4 (a). To simplify, we consider a cluster with one output and only two inputs $\{i_1, i_2\}$. We need three control variables per output, c_1 controlling the pin a of the cells, c_2 the pin b, and c_3 making the library cell selection. We consider that input i_1 has no other fanout in the circuit. The ADD of the capacitance at input i_1 is represented on Figure 4 (b). We see on this add when c_1 is 0 or (c_2 is 0) we select i_1 on the pin a (pin b), and when c_3 is 0 (1), we select AND2 (NAND2). For example, with the selection $c_1\bar{c}_2\bar{c}_3$, the capacitance at input i_1 is 2, because in this case, i_1 will have one single connection to pin b of AND2.

The ADD computing symbolically the power is given by the Equation 7. The first sum of the equation is the power consumed at the outputs of the cluster, the second sum at the inputs of the cluster.

$$\mathcal{P}(\mathbf{c}) = \sum_{o \in Out} P_t(\mathbf{c}, o)C(o) + \sum_{i \in In} P_t(i)(C_i(\mathbf{c})) \tag{7}$$

3.2 Bounding the Cost

Building the quotient function for the entire (large) library is expensive, and most of the time useless. Remember that we are performing re-mapping and re-optimization. It is not necessary to include in the quotient function cells and assignments for which we are certain that the global costs will be higher than those of the original implementation.

To suppress assignments that have a cost higher than the original implementation, we compute a power bound for each cell. This bound is the *minimum possible cost* that this cell has in any implementation in the cluster. If the bound is higher than the original cost in power, we discard the cell.

As for the cost function, we have to analyze two factors, the probability of transition at the outputs, and the capacitance at the inputs. The value of the probability of transition is not completely free at the outputs. The outputs are bound by the Boolean relation $\mathcal{F}(i, o)$. This relation defines a *range* on the transition probability at one of the outputs o_1. Given $\mathcal{F}$ and $P(\mathbf{i}, \mathbf{i}^+)$ it is possible to compute the minimum transition probability $P_{t_{min}}(o_1)$ at o_1 satisfying the constraint that the functions implementing o_1 is compatible with $\mathcal{F}$ (the same holds for o_2). We do not discuss this topic in detail because of space limitations [15]. $P_{t_{min}}$ is a bound on the transition probability at the outputs, regardless the cell representatives.

To compute an effective bound, we must account for the input capacitance as well. Without considering the functionality of the cell, we want to express a lower bound on the power consumed to drive the inputs of any cell in the library. We consider a cell g with inputs capacitances sorted in *decreasing* order $\{C_1, C_2, \ldots, C_n\}$. If we could connect the cell pins with any cluster input, the least input power would be consumed would be $C_1P_{t1} + C_2P_{t2} + C_3P_{t3} + \cdots$, where $\{P_{t1}, P_{t2}, \ldots, P_{tk}\}$ are the transition probabilities of the inputs sorted in *increasing* order. This is the *minimum possible* power consumed at the inputs of this cell for any input permutation.

We add to this bound the minimum power at both outputs. We do not know what is the other cell selected for the cluster, and therefore what is the power consumed at its inputs. So we sum to the bound the best case, usually an inverter because it has only one connection, and its power contribution is very low. If this bound is greater than the original power value, i.e. the power of the original implementation, we can discard the cell. We see here that the effectiveness of the bound depends highly of the quality of the original implementation. However, if the original implementation

is of bad quality, we can use a tighter value as the original value in a first pass, get a sub optimal solution, and run again by releasing the tighter value. Notice that our bound is very conservative, because we guarantee that no potentially optimal solution is eliminated. The bound discards on average 38% of the cells in the library, at a very low computational cost.

3.3 Timing constraints

Minimizing power is the optimization objective, but we need also to respect the timing constraints. Constraints can be manipulated in a symbolic fashion as well. Before describing their ADD-based representation, we describe how timing constraints are computed.

For each cluster output, arrival time and required time are computed. We can replace a cluster by an alternative implementation if the new arrival times at *all* cluster outputs do not exceed the required times. For timing constraints, we use the critical path of the circuit as the maximum delay acceptable from the primary inputs to the primary outputs. We compute the arrival and required times for all nodes by taking this constraint in account.

The ADD representing the arrival time at a node for all possible cell assignments and input pin assignments is computed using the following symbolic formula:

$$T_{arr}(\mathbf{c}) = \max_{i \in Inputs} (T_{arr_i}(\mathbf{c}) + \alpha(\mathbf{c}) \times C + \beta_i(\mathbf{c})) \qquad (8)$$

Where T_{arr_i}, α and β are ADDs in the control variables, and all operators involved in the computation are standard ADD operators. The leaves of T_{arr_i} contain all possible arrival times for the input i. The leaves of α contain all possible driving resistances of the cells. The leaves of β_i contain all possible intrinsic delays from input pins to the outputs of the cells. The algorithm for building the ADD for the timing constraint is similar to the one used for the construction of the power cost function. Once the ADD of the arrival times has been built, it can be used to prune the matching function. All the assignments of $\mathbf{c}$ that leads to leaves with value of the arrival time larger than the required violate the timing constraint and are discarded from the ON-set of $M(\mathbf{c})$.

4 The Power Estimation Engine

The computation of signal probabilities and transition activities for the entire network is performed using a Montecarlo approach [17] based on bit-parallel simulation [10]. The efficiency of BPS is high: we could simulate thousands of patterns for our largest benchmarks in a few seconds. Simulation time grows linearly with network size. Patterns or, alternatively, probabilities and transition activities can be specified for the inputs. The Montecarlo stopping criterion can be overridden by the user and full simulation of a pattern file can be performed.

Whenever the network is modified, the effect of the change must be evaluated. We call this step re-simulation. Re-simulation was implemented using BDD-based probabilistic techniques. The functionality of the network is unchanged outside the neighborhood of a cluster when a re-mapping has been performed. Hence, we can re-simulate the neighborhood alone, to check that the power saved in the re-mapping is not swamped by the effect of re-mapping on fanout gates within the neighborhood. Since the neighborhood is a small fraction of entire network, BDD-based probabilistic power estimation is performed in a very short time. It is important to notice that re-simulation does not take into account the correlation at the inputs of the neighborhood, thus it is not as accurate a global simulation in estimating the effects of a re-mapping (but it is much faster).

After implementing the re-simulation engine, our experiments revealed that the estimated power savings computed by the cost function were extremely close to those given by re-simulation. This conclusion is quite unexpected. It appears that the cost function is very accurate in estimating power savings and losses, and re-simulation is actually not needed. Although it is implemented and functional, it was not used in our runs. Nevertheless, to protect ourselves against pathological cases, we perform global BPS simulation every few re-mappings (usually 10). If power is increased the re-mappings can be undone. We conjecture that this event is extremely rare: it never happened in our tests.

5 Experimental Results

We have implemented a post-mapping power optimization tool based on generalized matching. The tool reads a mapped circuit described in `blif` (or `slif`) and a library file, and runs the optimization. Several user-controlled parameters can be specified. The depth of the neighborhood can range from 0 to infinity. The number of outputs of a cluster can be also controlled. We made experiments with up to four outputs. The number of inputs `i` of a cluster can be controlled as well. Usually they are assumed to be the inputs of the cells implementing the cluster in the original mapped netlist. However, additional inputs can be added taken from nodes in the neighborhood. With this simple modification, we can exploit the power of generalized matching to perform local re-wiring.

We can also change the cluster selection algorithm to select arbitrary sets of nodes as clusters. Experimentally, we observed that this is much less effective than starting from multiple fanout points, mainly because traditional logic optimization is already effective on fanout-free trees. Although some incremental optimization could be achieved (between 1% and 2% with respect with the results presented later), the run time cost was remarkable (between a factor of three and five).

Memory optimization is the primary concern in the software implementation. The tools uses the `Cudd` BDD package [18] which provides a rich set of operators on BDDs and ADDs and powerful memory management and caching features. We set up a memory limit of 1,000,000 BDD and ADD nodes. When this limit is reached, the matching is aborted and the traversal continues. Thus, when the BDDs exceed the memory limit the program simply frees the memory and moves on to new matchings.

We have tested our tool with a set of combinational MCNC benchmarks [19]. The benchmarks are optimized with SIS [12] for minimum delay with area recovery, with script `script.delay` followed by the mapping command `map -n 1 -AFG`. We used the critical path delay of the circuit synthesized for maximum speed as timing constraint for the optimization. Notice that we synthesized with area recovery because we wanted to

measure the effect of power optimization alone. Obviously, if the circuit is not area-minimal, some power reduction can be achieved just by reducing area, and this effect would inflate the power savings. Moreover, we could not compare fairly with the mapper for low power of POSE because it performs trade-offs between power and speed, and we could not guarantee that the tight delay constraints would have been satisfied.

Our tool was run with the same parameter settings on all benchmarks, in an effort to demonstrate robustness and generality. We ran the matching algorithm on clusters of two outputs, with the neighborhood search limited to a depth of 3. We used a library based on an industrial technology file, with 75 cells, with up to five inputs. The number of inputs of the cluster was limited to 10. Table 1 shows the benchmarks, their

Bench	Size	% Δ_A	% Δ_P	CPU
z4ml	48	0.32	13.77	75
b9	112	0.39	18.37	293
term1	180	-1.60	10.24	391
C432	183	-0.36	9.85	559
9symml	204	1.86	15.17	394
alu2	349	-0.19	13.97	1260
C499	365	4.25	26.82	1943
x4	365	-0.54	3.03	575
C880	379	0.51	9.99	1388
C1908	508	0.58	11.50	1503
C1355	524	-1.81	13.02	883
too_large	573	-0.79	11.30	1432
x3	640	-0.90	6.64	723
rot	677	-0.55	12.90	4519
apex6	694	-1.56	11.70	1126
alu4	699	0.26	16.07	3766
frg2	785	3.20	19.77	974
vda	785	3.46	24.25	3906
t481	863	-0.07	13.55	3779
dalu	966	0.29	18.39	5977
C2670	966	-0.06	12.08	1457
k2	1221	-1.83	7.42	4029
C3540	1352	-0.33	15.56	2971
pair	1484	-0.24	16.35	3739
C5315	2055	-1.40	13.97	20763
des	3621	-1.26	9.75	44842
C7552	3728	9.49	29.66	169634
C6288	4373	5.87	23.28	9470
Total		0.26	17.57	

Table 1: Results of power optimization on MCNC benchmarks.

gate count, the percentage area gain, the percentage power gain and the run time in seconds. We observe an average gain of more than 17% in power while the critical path of the circuit has been constrained to remain the same as the original circuit. The average is weighted: the power savings are weighted with the gate counts. Notice that the weighted average is higher than the arithmetic one. This can be explained by the fact that larger benchmarks have more MFPs and more internal degrees of freedom for mapping. It is important to notice that our power optimization is decoupled from area. In many cases we have marginal area savings, and the area losses are always within 2%.

The run times range from one minute to several hours for the biggest benchmark on a SGI Indy workstation with 96Mb of RAM, the exception being for one benchmark which ran for 48 hours. This is explained by the fact that the degrees of freedom for this benchmark were very big and the bound was less effective than for the other benchmarks (on the other hand, we achieved almost 30% power saving). These run times are of the same order than those spent by SIS in technology independent and technology dependent optimization. Most of the time is spend in building the matching function and in universal quantification of the variables in Equation 2.

6 Conclusions

In this paper we presented a novel technique for power optimization of mapped netlist under tight timing constraints. Our tool exploits the power of Boolean relations and symbolic cost function manipulation for finding power-optimal replacements for groups of cells. An essential component of the tool is a powerful and efficient power estimation engine based on bit-parallel simulation and incremental probabilistic estimation. We took special care in building an implementation that is robust enough to deal with large circuits in a reasonable time. As a result, we have presented (for the first time in the literature) power optimization results for the largest benchmarks in the MCNC91 suite.

Differently from numerous other approaches, we do not trade off speed for power. Our tool reduces power by 17% in average. Area is almost unchanged (0.26% reduction). The run time for re-mapping is of the same order as the time spent in optimization and mapping.

References

[1] M. Pedram, "Power minimization in IC design: principles and applications," *ACM TODAES* vol. 1, n. 1, pp. 1–54, 1996.

[2] C. Tsui et al., "Power efficient technology decomposition and mapping under an extended power consumption model," *IEEE TCAD*, vol. 13, n. 9, pp. 1110–1122, 1994.

[3] S. Iman et al, "Multi-level nework optimization for low power," in *ICCAD*, pp. 372–377, 1994.

[4] S. Iman et al., "Logic extraction and factorization for low power," in *DAC*, pp. 248–253, 1995.

[5] S. Iman et al., "POSE: Power optimization and synthesis environment," in *DAC*, pp. 21–26, 1996.

[6] B. Rohfleisch et al., "Reducing power dissipation after mapping by structural transformations," in *DAC*, pp. 789–794, 1996.

[7] R. Bahar et al., "Symbolic computation of logic implications for technology-dependent low-power synthesis," in *ISLPED*, pp. 163–168, 1996.

[8] Q. Wang et al., "Multi-level logic optimization for low power using logic transformations," in *ICCAD*, pp. 270–277, 1996.

[9] F. Somenzi et al., "Minimization of Boolean relations," in *ISCAS*, pp. 738–473, 1989.

[10] P. Schneider, "PAPSAS: A fast switching activity simulator," in *PATMOS*, pp. 350–360, 1995.

[11] L. Benini et al., "Generalized matching, a new approach to concurrent logic optimization and library binding," *ACM TODAES*, 1997.

[12] E. Sentovich et al., "Sequential Circuits Design Using Synthesis and Optimization," in *ICCD*, pp. 328-333, Oct. 1992.

[13] G. De Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill, 1994.

[14] R. Bahar et al., "Algebraic Decision Diagrams and their Applications," in *ICCAD*, pp. 188-191, 1993.

[15] P. Vuillod et al., "Generalized matching from theory to practice," *in preparation.*

[16] Y. Watanabe et al., "Permissible functions for multioutput components in combinational logic optimization," *IEEE TCAD* vol. 15, no. 7, pp. 734–744, 1996.

[17] R. Burch et al., "A Monte Carlo approach for power estimation," *IEEE TVLSI* vol. 1, n. 1, pp. 63–71, 1993.

[18] F. Somenzi. *The CUDD package User's guide. Version 1.0.5* 1995.

[19] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," *Technical report, MCNC*, Research Triangle Park, NC, 1991.