In-Place Delay Constrained Power Optimization Using Functional Symmetries

Chih-Wei (Jim) Chang, Bo Hu, and Malgorzata Marek-Sadowska
Department of Electrical and Computer Engineering,
University of California, Santa Barbara, CA 93106, USA

Abstract

In-Place Optimization (IPO) has become the backend methodology of choice to resolve the gap between logic synthesis and physical design as the optimization can be guided by accurate physical information. To perform optimization without perturbing too much the placed netlist, only buffer insertion and gate sizing are commonly used in current design tools. In this paper, we address the problem of delay-constrained power optimization by introducing another degree of freedom: functional symmetry based rewiring. Theoretical results on the effect of using functional symmetry on transition density for power estimation is also derived. Experimental results show that, under the same delay constraint, our technique achieves much better power reduction as compared to the discrete gate sizing only technique.

1. Introduction

Power consumption and speed are two primary cost functions in today's integrated circuit design. As mobile computation devices prevail in the market, the ability to design faster, low-power devices is of paramount importance. However, these two objectives are often conflicting: faster circuit consumes more power while low-power circuit runs slower. Hence, designers often need to trade-off power for speed and vice versa to meet the desired specification. To get the best performance, power and speed are considered at various stages of the design cycle, including architecture, RTL, gate, and layout levels. Due to the migration of IC processes into finer feature sizes, interconnect induced delay could easily dominate the critical paths and hence delay estimation made during logic synthesis could be very inaccurate. In this paper, we target the power-delay trade-off specifically at the post-placement level when the gate level netlist is already placed on a 2-dimensional plane. The rationale behind this methodology is that delay information can be estimated accurately based on the placement solution such that power-delay trade-off can be done more effectively.

Since the placement is already done, optimization techniques used at this level should not perturb the existing placement solution too much in order to guarantee timing closure. Buffer insertion and gate sizing are traditionally the only two techniques that are suitable for this purpose. However, these two techniques are limited to the existing netlist without being able to explore a larger solution space by restructuring the logic. In recent years, rewiring techniques, such as redundancy-addition-and-removal[6] and functional symmetry based technique[2], have been successfully applied at post-layout to restructure the logic. These techniques have the property that only wires are reconnected such that the existing placement solution can be left intact. This property is very important as it allows logic changes to be guided by accurate delay information.

To have a quick feedback about the quality of the synthesis results, probability-based power estimation techniques are commonly used for their computational efficiency. [7] gives an excellent review of various such estimation techniques. Many techniques have been proposed in the past to minimize power consumption at logic synthesis level. In [5], a unified framework for low-power design is proposed. Rewiring techniques are used for power optimization in [3][9][12]. Basically, rewiring is used to change the toggle rate of internal signals such that gates with large load capacitance are assigned lower toggle rate by rewiring. Since application of rewiring can potentially change the toggle rate of gates in the whole circuit, each rewiring decision is made based on its effect on power reduction and greedy algorithms are used. Delay constraint are either neglected[3] or roughly estimated[9][12].

In this paper, we present a delay-constrained power optimization algorithm based on the notion of generalized implication supergate rewiring[2]. This type of rewiring has the property that the transition densities [8] at the roots of the supergates remain unchanged when wires are reconnected. This enables us to design a global optimization algorithm tightly coupled with traditional discrete gate sizing for a careful and accurate power-delay trade-off exploration.

2. Preliminaries

The average power dissipation in a CMOS gate consists of three major factors:

$$P_{av} = P_{load} + P_{short} + P_{leak}$$

The first term, P_{load}, is the power consumed when charging and discharging the output load of the gate. It depends on the output loading capacitance and the toggle rate (number of transitions per time unit). The second term, P_{short}, is due to the short circuit current during the CMOS gate's switching. It depends on the input transition time, internal load, and the toggle rate. The last term P_{leak} is the power consumed due to the device leakage current. Since P_{short} and P_{leak} are more device related, we only consider the optimization of P_{load}, which is the dominating factor of P [7].

The toggle rate depends on the relative delays of signals propagating through the circuit. A gate can undergo a series of transitions before settling to a steady state. However, it is computationally very expensive to determine this effect as it involves an event-driven simulator with all timing information considered[7]. In order to use the estimation as a

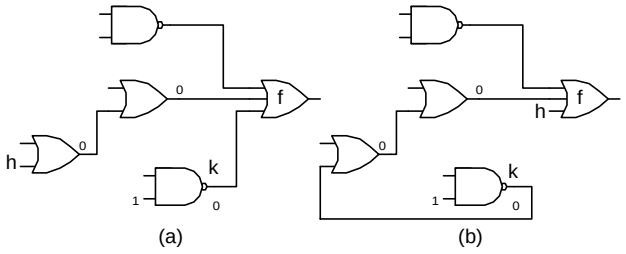


Fig. 1: h and k are swappable

sub-routine inside our algorithm, we choose to neglect the effect of glitching and use a zero-delay model instead.

Najm has introduced the notion of equilibrium probability and transition density for power estimation[8]. The equilibrium probability of a signal *x*, denoted P(*x*), is the fraction of time *x* is evaluated to logic 1. The transition density of *x*, denoted D(*x*), is the average number of transitions per unit time. Under spatial and temporal independence assumption, an efficient algorithm was introduced to propagate the density values from the primary inputs throughout the circuit. To see how the propagation algorithm works, recall the concept of Boolean difference: if *f* is a Boolean function that depends on *x*, then the Boolean difference of *f* with respect to *x* is defined as:

$$\frac{\partial f}{\partial x} = f|_{x=0} \oplus f|_{x=1} = f_{\bar{x}} \oplus f_x$$

Here, $\oplus$ represents the Boolean exclusive-or function. The Boolean difference is the XOR of the positive and negative cofactors with respect to *x*. Essentially, $\frac{\partial f}{\partial x}$ is the condition that if there is a transition on *x*, there is a corresponding transition on *f*. For example, let *f* be a two-input AND gate. i.e. $f = x_1 \cdot x_2$. The Boolean difference of *f* with respect to x_1 is $\frac{\partial f}{\partial x_1} = 0 \oplus x_2 = x_2$. So, when $x_2 = 1$, any transition at x_1 will cause a corresponding transition at *f*.

It is shown[8] that under the spatial independence assumption, the transition density at the output of a *n*-input function *f* can be calculated by the following equation:

$$D(f) = \sum_{i=1}^{n} P\left(\frac{\partial f}{\partial x_i}\right) D(x_i)$$

Intuitively, $D(f)$ is the summation of each of the inputs' transition densities multiplied by the probability of setting other side inputs for the propagation of the transition. The overall power consumption estimation under this measure is then:

$$P_{av} = \frac{1}{2} V_{dd}^2 \sum_{i=1}^{k} C(x_i) D(x_i)$$

where V_{dd} is the supply voltage and $C(x_i)$ is the load capacitance seen from node x_i. *k* is the total number of nodes in the circuit.

Let f: $\{0, 1\}^n \rightarrow \{0, 1\}$ be a single-output completely specified Boolean function defined on the input set X = $\{x_0, x_1, ..., x_{n-1}\}$. Four cofactors can be defined with respect to two variables x_i, $x_j \in X$. i.e. $f_{\overline{xi}\,\overline{xj}}$, $f_{\overline{xi}\,xj}$, $f_{xi\,\overline{xj}}$, and $f_{xi\,xj}$. *Symmetry* is defined as the equivalence between any of the two cofactors. Among them, two types of symmetries are of special interest and are stated below:

Definition 1: x_i and x_j are *non-equivalence symmetric* (NES) in *f*(X) if and only if $f_{\overline{xi}\,xj} = f_{xi\,\overline{xj}}$. That is, the exchange of x_i and x_j does not change *f*. i.e. $f(..., x_i, ..., x_j, ...) = f(..., x_j, ..., x_i, ...)$.

Definition 2: x_i and x_j are *equivalence symmetric* (ES) in *f*(X) if and only if $f_{\overline{xi}\,\overline{xj}} = f_{xi\,xj}$. That is, the exchange of $\overline{x}_i$ and $\overline{x}_j$ does not change *f*. i.e. $f(..., x_i, ..., x_j, ...) = f(..., \overline{x}_j, ..., \overline{x}_i, ...)$.

We now discuss the concept of implication supergate and its relationship to functional symmetry. Implication supergate was introduced in [11] for Automatic Test Pattern Generation (ATPG). It was later generalized[2] and used to identify easily detectable functional symmetries in a Boolean network[1]. The ability for delay optimization was demonstrated in [2]. Generalized implication supergate is a set of connected gates that functionally behave like a big AND, OR, or XOR gate. For example, the circuit in Fig. 1(a) consists of five gates but they behave functionally the same as a big OR gate rooted at *f* with some input phase inversions. To extract all generalized implication supergates from a given netlist, we start from the primary outputs and process each gate in a reverse topological order. At each primary output, depending on its gate type, either direct backward implication or xor propagation is attempted. Multiple-fanout nodes, or nodes where backward propagation stops, are treated as new implication supergate roots and the propagation process continues. This procedure stops when all primary inputs are reached. After the extraction, the network is uniquely partitioned into AND, OR, and XOR supergates with inverters and buffers at their pins.

The most important result developed in [2] is that wires that are covered by the same generalized implication supergate are functionally symmetric and can be swapped without changing the overall functionality of the circuit. This is illustrated in Fig. 1. Since pin *h* and *k* are covered by the same implication supergate rooted at *f*, they are functionally symmetric and can be swapped. The swap results in netlist shown in Fig. 1(b) which is functionally equivalent to the one in Fig. 1(a). Wire swapping has two major effects on timing optimization. First, it may reduce the number of levels the critical path has to travel. For example, if the critical path extends from *k* to *f* in Fig. 1(b), then it is beneficial to swap the wires to achieve the result in Fig. 1(a). Second, the interconnect loading can dramatically be reduced on the critical path. The reader is referred to [2] for more details.

3. In-Place Delay Constrained Power Optimization

Power and delay are two conflicting factors in design trade-off. Usually, the designer is willing to trade a prespecified delay penalty for as much power reduction as possible. In this section, we discuss our approach to this

problem. We start by analyzing the effect of wire swapping on transition density.

3.1 Effect of Swapping on Transition Density

Theorem 1: Let f be a function defined over support set X = $\{x_1, x_2,..., x_n\}$ and f is of NES (ES) with respect to variables x_i, $x_j \in X$. Let $D^{new}(f)$ be the transition density at f after swapping x_i and x_j. Then, the transition density after the swap is equal to the transition density before the swap. That is, $D^{new}(f) = D(f)$.

proof:

Without loss of generality, we assume f is of NES with respect to variables x_i, x_j. That is:

$$f_{\bar{x}_i x_j} = f_{x_i \bar{x}_j} \qquad (1)$$

The case for ES can be proved similarly. The swap is illustrated in Fig. 2.

By definition, the transition density of f before swap is

$$D(f) = \sum_{k=1\ldots n} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k)$$
$$= \sum_{\substack{k=1\ldots n \\ k \neq i,j}} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k) + P\left(\frac{\partial f}{\partial x_i}\right) D(x_i) + P\left(\frac{\partial f}{\partial x_j}\right) D(x_j)$$

For simplicity, we denote $D_t(f)$ as the last two terms of $D(f)$. That is, $D_t(f) = P\left(\frac{\partial f}{\partial x_i}\right) D(x_i) + P\left(\frac{\partial f}{\partial x_j}\right) D(x_j)$.

The new transition density of f after the swap is:

$$D^{new}(f) = \sum_{k=1\ldots n} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k)$$
$$= \sum_{\substack{k=1\ldots n \\ k \neq i,j}} P\left(\frac{\partial f}{\partial x_k}\right) D(x_k) + P\left(\frac{\partial f}{\partial x_i}\right) D(x_j) + P\left(\frac{\partial f}{\partial x_j}\right) D(x_i)$$

We denote the last two terms of $D^{new}(f)$ as $D_t^{new}(f) = P\left(\frac{\partial f}{\partial x_i}\right) D(x_j) + P\left(\frac{\partial f}{\partial x_j}\right) D(x_i)$. It is to be noted that $P\left(\frac{\partial f}{\partial x_i}\right)$ is now associated with $D(x_j)$. It is clear that $D(f) = D^{new}(f)$ if and only if $D_t(f) = D_t^{new}(f)$. Now we proceed by expanding $\frac{\partial f}{\partial x_i}$ and $\frac{\partial f}{\partial x_j}$

$$\frac{\partial f}{\partial x_i} = f_{x_i} \oplus f_{\bar{x}_i}$$
$$= (f_{x_i} \oplus f_{\bar{x}_i})\big|_{x_j} + (f_{x_i} \oplus f_{\bar{x}_i})\big|_{\bar{x}_j} \qquad \text{(Shannon Expansion)}$$
$$= (f_{x_i x_j} \oplus f_{\bar{x}_i x_j}) + (f_{x_i \bar{x}_j} \oplus f_{\bar{x}_i \bar{x}_j}) \qquad (2)$$

$$\frac{\partial f}{\partial x_j} = f_{x_j} \oplus f_{\bar{x}_j}$$
$$= (f_{x_j} \oplus f_{\bar{x}_j})\big|_{x_i} + (f_{x_j} \oplus f_{\bar{x}_j})\big|_{\bar{x}_i} \qquad \text{(Shannon Expansion)}$$
$$= (f_{x_i x_j} \oplus f_{x_i \bar{x}_j}) + (f_{\bar{x}_i x_j} \oplus f_{\bar{x}_i \bar{x}_j}) \qquad (3)$$

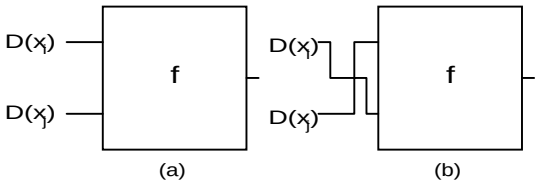


Fig. 2: Swap effect on transition density (a)before swap, (b) after swap

By assumption, x_i and x_j are of NES and equation (1) holds. Plugging in (1) into Equation (2) and (3), we obtain $\frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial x_j}$. This result in turn proves the equivalence between $D_t(f)$ and $D_t^{new}(f)$. Finally, we conclude that the new transition density after the swap is the same as the one before the swap. QED.

The importance of Theorem 1 is two fold. First, it provides the theoretical fundation for the effect of symmetric swapping on transition density. Changes in transition density are guaranteed to be bounded inside the associated implication supergate. Second, transition densities at each implication supergate roots serve as a set of fixed points throughout the optimization. As a result, our algorithm can have a global view of the whole optimization problem. Detail algorithm will be discussed in the next section.

3.2 Coupling Gate Sizing with Functional Symmetry

Coudert has proposed a discrete gate sizing algorithm in [4]. The idea is to interleave the conventional greedy approach with a relaxation phase in order to jump out of local optima. Given a gate $g \in N$ in the netlist, each implementation from the library that has the same functionality of g is viewed as a possible move to be executed. Since power consumed by a gate is the product of its toggle rate and load capacitance. Gate resizing can effectively reduce the loading at high switching gate. Moves are graded based on their *fitness*, the effect of executing the move on local neighborhood. The slack at each gate g is defined as the difference between its required time and arrival time.

Our approach is an extension to the aforementioned algorithm by considering not only gate resizing, but also wire swapping. Wire swapping contributions to delay constrained power optimization are: 1) The transition density of gates covered by the same supergate can potentially be changed. Thus, it is beneficial to lower the transition density at gates with high loading by wire swapping. 2) Gate resizing lowers the power consumption at the cost of delay penalty. When the allowable delay penalty is reached, no further power reduction is possible. The room for trade-off can be enlarged by covering the delay loss with wire swapping, which has been shown to be good for delay optimization[2]. To tightly couple these two choices, we observed that a given netlist can be viewed as a netlist of interconnected supergates after supergate extraction. Each possible swap of a supergate can be viewed as electrically different while functionally equivalent implementation from a virtual library of this supergate. For supergates that are nontrivial (a supergate is trivial if it covers only one gate), we consider each swap as a possible move. For trivial super-

```
function gsg_sizing(N: input netlist, L: technology library)
update ← all gsg in the circuit;
moves ← ∅;
loop
  old_cost ← Cost(N);
  foreach gsg ∈ update
    gsg.move ← ∅;
    gsg.fit ← Fitness(N);
    if gsg is trivial
      foreach gates g ∈ L implementing gsg
        fit ← Fitness(N[gsg ← g]);
        if (fit > gsg.fit)
          gsg.fit ← fit;
          gsg.move ← g;
    else {//non-trivial gsg
      foreach swap s of gsg
        fit ← Fitness(N[gsg ← gsg(s)];
        if (fit > gsg.fit)
          gsg.fit ← fit;
          gsg.move ← s;
  //end of "foreach gsg ∈ update"
  moved ← BestMultipleMoves(N);
  update ← GetPerturbedNodes(N, moved);
until Converge(old_cost, Cost(N), moved)
```

Fig. 3: Algorithm

gate, each implementation of this gate from the technology library forms the set of possible moves. Hence, a move in our algorithm can be either resizing a gate or swapping of wires.

Now we analyze the effect of each type of moves. Basically, the resizing will affect the slack(ΔS) and power(ΔP) of the circuit under optimization. In [4], Coudert has observed that the effect on slack tends to be confined within the local neighborhood of the move as the effect of resizing a gate was analyzed. Here we concentrate on the effect of a swap. The change in slack can be calculated by updating the arrival/required time in the local neighborhood. The change in power consumption comes from two sources: 1). The loading capacitance of the swapped pins are changed. 2). The transition density of the fanouts of the swapped pins are changed up to the root of the supergate. This effect can be efficiently calculated by an event-driven procedure.

We adopted a benefit/penalty function based approach for the delay constrained power optimization problem by defining the fitness function of each move as follows:

$$\text{Fitness} = \begin{cases} 0 & \Delta S < 0, \Delta P > 0 \\ e^{\alpha \Delta S + \beta \Delta P} & \text{otherwise} \end{cases}$$

where ΔS is the change in minimum slack in the local neighborhood and ΔP is the change of the power consumption of the whole circuit. α and β are pre-defined constants. A move is assigned zero fitness value(gain) if the move causes both the slack and power to be worse. Otherwise, the gain is defined as a function depending on both ΔS and ΔP. In general, we want to choose a move that trades as little ΔS to achieve as much ΔP as possible. A move can be either a gate resizing or a wire swapping and are distinguished only by their fitness values.

The overall algorithm is shown in Fig. 3. For each implication supergate *gsg* in the netlist, we find the best move based on the fitness value calculated over the local neighborhood. For *gsg* that is trivial, we consider resizing as the set of the candidate moves. After finding best moves for each *gsg*, the algorithm sorts all the *gsg* into sequence with respect to their fitness values. A series of best moves is determined by traversing the sequence of moves. It is to be noted that we do not stop at the first maximum found when traversing the sequence. Instead, we traverse the whole sequence and determine the best sequence in order to escape from local optima. This is implemented in the BestMultipleMoves(N) function. After applying the moves, the set of gates that are in the neighborhood of the perturbed gates are put into the *update* list as the candidates for next iteration. The algorithm stops when convergence condition is met, either the iteration limit exceeds or the improvement is lower than a given threshold.

3.3 Interconnect Modeling

Since the final routing is not available after placement, a net model is necessary to estimate the delay along the interconnect. Assume all pins have known coordinates after placement. Each net is modeled as a star: the center of the star is the center of gravity of all its terminals. A net is divided into several segments: from source to the star center and from the star center to each sinks. Each segment is modeled by lumped RC. We use Elmore delay model for delay calculation. Since the distance from the star center to each sinks may vary, each sink may have different delay from the source.

On the gate delay side, we use a load-dependent model. The delay from an input pin i to an output pin g is

$$\delta(i, g) = \alpha_{i,g} + \beta_{i,g} c_g$$

Here, c_g is the load capacitance at the output of gate g. $\alpha_{i,g}$ is the intrinsic delay from in-pin *i* to the out-pin of *g*. $\beta_{i,g}$ is the load dependent coefficient. Each α and β have two values corresponding to the rise and fall transitions respectively.

4. Experimental Results

Our prototype tool, RAPIDS-P (Rewiring After Placement usIng easily Detectable Symmetries for Power optimization), has been implemented on top of SIS 1.3[10] and tested on both MCNC 91 and ISCAS 89 benchmark suites. Sequential circuits are treated as combinational ones with all sequential elements removed. All benchmarks are optimized by SIS *script.rugged* and mapped by command "map -n 1 -AFG". We use a commercial 0.35μm standard cell library consisting of INV, BUF, NAND, NOR, XOR, and XNOR with number of inputs ranging from 2 to 4. Each type has 4 different implementations. The mapped netlist is fed to a commercial timing-driven placer. Cell locations are extracted after placement. To model interconnect, we use 2 pf/cm for unit capacitance and 2.4KΩ/cm for unit resistance. Transition density and equilibrium probability are assigned 100 and 0.5 at all primary inputs for all benchmarks. Benchmarks runs are performed on a Pentium III 500MHz PC.

TABLE 1. Experimental results

	initial circuit				fixed delay constraint		fixed power constraint		area change		CPU	
					gs	gsg+gs	gs	gsg+gs	gs	gsg+gs	gs	gsg+gs
	# g	D	P	A	%P	%P	%D	%D	%A	%A		
alu2	516	9.1	2104	56526	8.1	12.1	7.7	-1.3	1.1	0.6	14	26
alu4	1004	12.6	3844	5111253	10.7	12.5	2.3	0.2	0.6	1.1	27	55
C432	291	9.8	1194	31573	7.5	9.3	2.9	-0.9	-0.7	-0.8	4	7
C499	625	5.8	4740	69992	10.3	15.6	3.8	2.4	-3.6	-4.0	12	28
C1355	625	5.7	4855	70495	4.6	5.0	10.5	7.5	-4.2	-1.7	14	19
C1908	730	9.4	3536	79411	8.2	11.3	8.1	5.5	0.5	1.0	18	21
C2670	911	6.9	5257	101777	11.1	13.8	4.5	1.9	-1.2	0.4	23	35
C3540	1809	12.7	11559	204197	6.3	13.5	13.2	3.1	0.1	0.5	53	93
C5315	2379	9.5	16158	267644	7.2	15.9	6.7	2.1	-4.1	-3.6	45	79
C6288	5000	39.1	167898	584906	7.1	25.5	7.0	1.2	-10.9	-10.1	158	143
C7552	2565	9.8	15577	286032	6.8	12.5	7.5	3.3	-2.3	-1.9	48	71
k2	1484	6.7	3087	164329	6.6	11.9	12.5	0.3	-0.2	0.5	30	74
i8	1229	6.4	6802	139671	6.6	10.0	NA†	5.9	0.8	0.0	101	163
i10	3397	17.4	23341	384827	13.1	14.5	0.7	-7.9	-2.7	-2.6	70	125
s13207	2900	10.7	13291	329798	11.4	12.7	2.1	0.1	-4.3	-3.7	44	106
s15850	4640	12.8	28248	536498	10.3	11.9	1.8	0.9	-4.7	-4.4	124	288
s38417	10090	15.1	82094	1139460	5.3	6.0	0.1	0.0	-2.0	-1.6	732	1420
average					8.3	12.6	5.7	1.4	-2.2	-1.8		

† gs-only is unable to reach 10% power reduction.

Table 1 shows the results. The first column lists the name of each benchmark. Column 2 shows the number of gates in the mapped netlist. To have a fair comparison with the gate sizing only technique, we preprocess the circuit by minimizing the critical path delay using only gate sizing. Column 3, 4, and 5 are the corresponding delay, power, and area after timing optimization. Column 6 and 7 are the corresponding power reduction for the gate sizing only approach and our hybrid approach when the delay constraint is set at 5% of the preprocessed circuit. To demonstrate the result from another angle, we set the power constraint to be 10% less than the preprocessed circuit and show the delay trade-off. The result for both approaches is at column 8 and 9, respectively. We also show the percentage of area perturbation and CPU time (in second) when deriving the power-delay trade-off curve from column 10 to 13.

The results clearly show the benefit of using functional symmetry together with gate sizing for post-placement power-delay trade-off. In all benchmark runs, the hybrid approach always reaches better power reduction at less cost of delay penalty. For example, in benchmark C6288, the gate sizing only approach reduces power by 7.1% at 5% of delay penalty. At the same delay penalty, the hybrid approach reaches as much as 25.5% reduction in power consumption. On the other hand, delay penalty of 7.0% and 1.2% for the gate sizing only and hybrid approach when the same benchmark is reduced to 90% of its original power consumption. This shows the great potential of our approach to trade less delay penalty for better power reduction. On average, at 5% delay penalty, our hybrid approach achieves 12.6% power reduction as compared to 8.3% of the gate sizing only approach. At 10% power reduction, we trade in only 1.4% of delay while using gate sizing only need 5.7% delay penalty. It is to be noted that in our experiment, only trivial supergates are considered to be resized. Further improvement is still possible by relaxing this constraint.

Our approach can potentially explore a much larger solution space than can be obtained by the gate sizing only approach. This can be seen in the power-delay trade-off curves in Fig. 4. It is easily seen from the curves that our hybrid approach can quickly reach a significant power reduction while trading in very small delay penalty. In Fig. 4(d), the processed benchmark alu2 has power level at 2104 and delay at 9.07. Our hybrid approach immediately finds a solution with delay 8.95 with the same power consumption. This shows that because we have a much larger solution space, we can have much more freedom in trading less delay for more power reduction.

5. Conclusion and Future Work

We have presented a combined rewiring and gate sizing technique for post-placement delay-constrained power optimization. Theoretical results on the use of functional symmetry and its effect on transition density are formally stated. With the set of implication supergate roots serving as fixed transition density points during the logic restructuring, we have developed a restructuring approach having a much more global view than existing greedy restructuring approach. Our technique can be distinguished from the existing techniques in several aspects: 1) Instead of trying to globally change the transition density of the circuit, it keeps a set of fixed transition density points in the circuit. This enables wire swapping to cover the delay loss when optimizing for power in a global fashion. 2) Performing optimization at post-placement stage allows us to accurately model the interconnect induced delay and carefully trade it for power. It is also to be noted that even though we use transition density based on [8] as our primary means for power estimation, our approach is not limited to it.

Other estimation techniques, such as simulation or symbolic techniques can be used for better accuracy. The interconnect model we adopted can be viewed as "perfect" because no non-overlapping or congestion constraints are considered. The performance improvement achieved under this model serves as a lower bound for more accurate interconnect models.

Experimental results show that our technique achieves much better power-delay trade-off compared to the gate sizing only approach. At post-layout stage, trading as little delay penalty as possible for large power reduction is very important as any delay penalty might lead to failed performance target. Our logic restructuring technique serves as a very good candidate to link the gap between logic synthesis and physical design.

Acknowledgments

This research was sponsored in part by Semiconductor Research Corporation grant 98-DJ-619 and in part by the National Science Fundation grant CCR 9811528.

References

[1] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis", Proc. IEEE, vol. 78, pp.264-300, Feb. 1990.

[2] C. -W. Chang, C. -K. Cheng, P. Suaris, and M. Marek-Sadowska, "Fast Post-placement Rewiring Using Easily Detectable Functional Symmetries", pp. 286-289, Design Automation Conference, 2000

[3] K. -S. Chung and C. L. Liu, "Local Transformation Techniques for Multi-Level Logic Circuits Utilizing Circuit Symmetries for Power Reduction", ISLPED 98, pp. 215-220

[4] O. Coudert, "Gate Sizing for Constrained Delay/Power/Area Optimization", in IEEE Trans. on VLSI, pp. 465-472, Dec. 1997

[5] S. Iman and M. Pedram, "POSE: Power Optimization and Synthesis Environment", pp. 21-26, Design Automation Conference, 1996

[6] Y. -M. Jiang, A. Krstic, K. -T. Cheng, and M. Marek-Sadowska, "Post-Layout Logic Restructuring for Performance Optimization", in Proc. of DAC, pp. 662-665, 1997

[7] F.N. Najm, "A survey of power estimation techniques in VLSI circuits". IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.2, (no.4), Dec. 1994. p.446-55

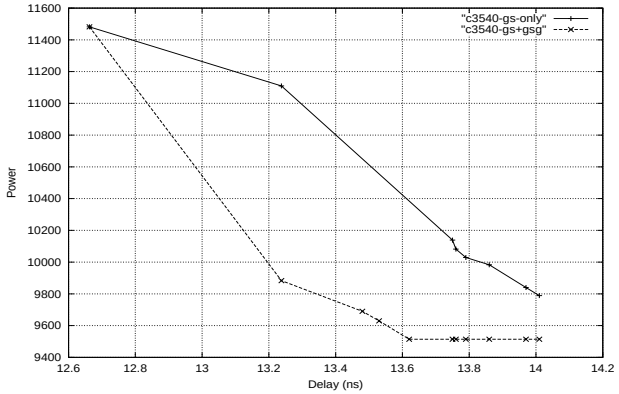
[8] F. N. Najm, "Transition Density: A New Measure of Activity in Digital Circuits", IEEE Transactions on Computer-Aided Design, vol. 12, no. 2, Feb, 1993, pp. 310-323

[9] B. Rohfleisch, A. Kolbl, and B. Wurth, "Reducing Power Dissipation after Technology Mapping by Structural Transformations", pp. 789-794, Design Automation Conference, 1996

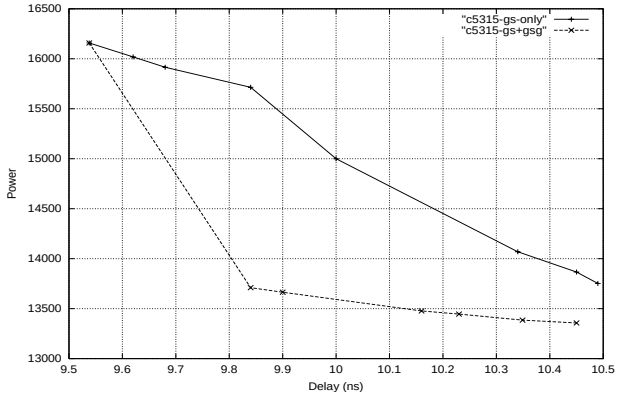
[10]"SIS: A System for Sequential Circuit Synthesis", Report M92/41, University of California, Berkeley, May, 1992

[11]K.-H. Tsai, R. Tompson, J. Rajski, and M. Marek-Sadowska, "STAR-ATPG: a high speed test pattern generator for large scan designs", Proceedings of International Test Conference 1999, pp. 1021-1030

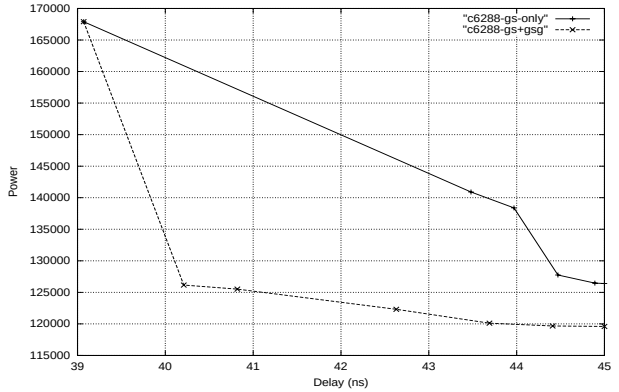
[12]Q. Wang, S.B.K. Vrudhula, G. Yeap, and S. Ganguly, "Power Reduction and Power-Delay Trade-Offs Using Logic Transformations", ACM Trans. on Design Automation of Electronic Systems, Vol. 4, No. 1, January, 1999, pp. 97-121



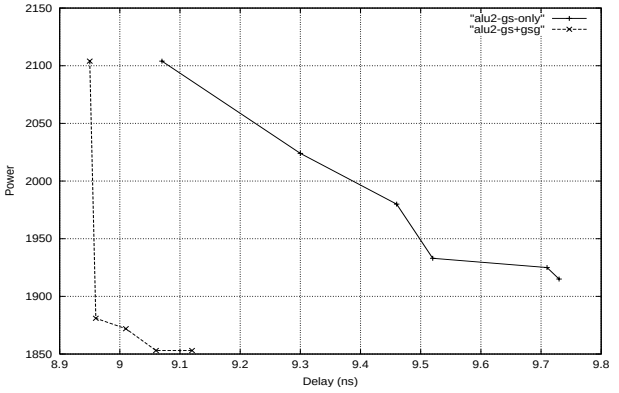
(a)C3540



(b)C5315



(c) C6288



(d) alu2

Fig. 4: Power-delay trade-off curves for four benchmark circuits