Timing-aware power noise reduction in placement

Chao-Yang Yeh and Malgorzata Marek-Sadowska
University of California, Santa Barbara
IBM Technical Contacts: Frank Liu and Sani Nassif
IBM Austin

Abstract

In this paper, we describe a placement-level decap insertion technique whose objective is to reduce power-noise, taking into account circuit timing. Our approach consists of prediction and correction steps. Before placement, we estimate the power noise of each cell considering switching frequency of cells which, after placement, will most likely be in the neighborhood. If a frequently switching cell has neighbors that switch infrequently, it is unlikely that this cell will suffer from a power noise problem. Based on the cell power noise estimation, we add decap padding to each cell. Then we invoke a standard cell placement tool and perform power grid analysis. We eliminate the power grid noise by gate sizing. Our technique can allocate decaps to improve power noise, power consumption, and timing. We propose two gate-sizing algorithms. The first one uses a Sequence of Linear Programs (SLP) formulation, and the second one uses a budgeting-based heuristic algorithm. The SLP algorithm can produce better power noise results than the heuristic, at the expense of run time. Experimental results show that our techniques can effectively reduce power noise and still meet timing constraints.

1. INTRODUCTION

Modern designs manufactured in advanced technologies are very sensitive to power noise. Aggressive technology scaling increases average current density and power noise magnitude. Reduced supply voltage causes power voltage drop to consume an increased portion of the ideal voltage supply level, which affects timing of CMOS gates. It is therefore important to address timing issues related to power noise.

Decoupling capacitance (decap) insertion is an effective way to reduce power noise. Decaps are intentionally inserted in the layout and attached to the power grid. Decap locations are important to ensure effectiveness in reducing power noise, so it is usually desirable to move them closer to the noisy areas.

In [5][6][14][19] decap allocation optimization is addressed at the floorplan level. In [5], the authors use iterative transient analysis and optimize decap locations. In [6], the authors formulate the decap placement as a network flow optimization problem. In [14], the authors distribute decaps proportionally to the values of currents drawn in each region. In [19], the authors observe that an effective way to allocate decaps is to distribute them to all grid nodes, assigning more decaps to grid nodes of the blocks with high switching rates. Some previous works [4][13] propose to reduce power noise by spreading the frequently switching cells evenly across the chip to eliminate hot spots. In [4], the authors include thermal cost function in a partition-based placer. In [13], the authors modify a quadratic placer to optimize both total power consumption and heat dissipation. Post-layout decap reallocation algorithms are proposed in [15][16]. Both [15][16] use power-noise sensitivity analysis to decide decap locations in the layout. In [16], the authors compute the sensitivity and conduct decap re-allocation only once. In [15], the authors compute sensitivity and move decaps many times for further improvement.

If in a certain area after the initial placement the power noise is severe, significant decap re-allocation is required. However, drastic changes of decap locations after placement should be avoided because timing, wire length, and other circuit properties might be significantly changed. Combining decap allocation with placement increases the number of placeable objects, which in turn increases the complexity of placement. The quality of decap allocation will also be seriously impacted by the early placement partition decision which usually relies on incomplete layout information. No previous works on decap allocation have considered timing, even though voltage drop may seriously impact a chip's timing.

In this paper, we address the decap allocation problem at the placement level. The floorplanner distributes the available decaps among the macro-blocks. Our goal is to find the final locations for decaps inside the individual blocks. We propose a timing-aware power-noise reduction scheme consisting of prediction-based decap allocation and gate-sizing algorithms. The flow of our noise-reduction methodology is shown in Figure 1. First we execute the *prediction* step. The goal of this step is to select the right amount of decap to be placed in the neighborhood of a cell. For each cell, prior to placement, we predict the size of the required decap and pad the cell accordingly (as shown in Figure 2). The better we can predict power-noise-affected cells before placement, the fewer decap re-allocations will be required after placement, and the better use we can make of the available decap area. The decap size prediction is based on the cell's current consumption and the expected placed-cell neighborhood. If a cell has high current consumption and its placement neighbors also have high current consumption, it is likely that this cell will suffer from excessive power noise. It will be less accurate to predict this cell's need for decap based only on its switching while

ignoring its neighbors. We predict a cell's neighborhood based on the wire length prediction and circuit structure analysis. Mutual contraction is utilized as the wire length prediction metric [7]. Previous work on wire length prediction will be explained in later sections. Although we focus on cell-level decap padding in this paper, our prediction-based padding method can also be applied to mixed-size or macro-cells netlists.

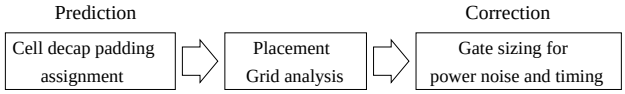


Figure 1. Prediction and correction power-noise reduction

Figure 2. Decap padding

After the cell padding, we perform placement followed by the power grid analysis to obtain new circuit delay information. The second optimization step is *correction*. We propose gate-sizing algorithms to improve power noise, power consumption, and timing after placement. Cell power noise is not only affected by placement of its neighbors, but also greatly influenced by the grid design and power pad location. However, these factors are not easily predictable. We need a gate-sizing step to help us meet power noise and timing goals. Our gate-sizing algorithms also consider decap-location optimization. Because the total chip area is fixed, if a gate area is changed, the decap area will be changed accordingly. We need to consider gate-sizing and decap-location optimization together.

We propose two new gate-sizing algorithms. The first algorithm linearizes the original non-linear expressions for gate-delay calculation and uses a Sequence-of-Linear-Programs (SLP)-based gate-sizing approach. The optimization is done by solving a linear program (LP) in each optimization iteration. The second gate-sizing algorithm is an iterative budgeting-based heuristic. In each iteration, cell sizes are adjusted in a way that no timing violation occurs. The heuristic algorithm can achieve results close to those of the SLP method; however, the runtime is much smaller. In our gate-sizing algorithms, we do not compute noise sensitivity as in [15][16], because we include the grid simulation in the optimization process. The voltage-drop simulation results are used to measure cell power noise sensitivity.

The contributions of this work are as follows. We point out that decap assignment should not be limited only to in-placement and post-layout optimization. Pre-layout decap prediction can significantly improve the results. We derive gate-sizing algorithms that take into account decap allocation and timing. Experimental results show that our power noise reduction techniques are effective. The gate-sizing formulation is also applicable to reducing power consumption and meeting timing constraints.

This paper is organized as follows. In Section 2, we show the background for modeling power grid, quantifying power noise, and predicting wire length. In Section 3, we discuss decap prediction and cell padding. In Section 4, we describe the gate-sizing correction process. In Section 5, we show the experimental results. We conclude the paper in Section 6.

2. BACKGROUND

In this section, we describe the models used in this paper. We also explain the mutual-contraction metric, which is used for pre-layout wire length prediction.

2.1 Modeling power grid, decap, cell delay and power noise measurement metrics

Power grid can be modeled as a mesh composed of resistors, capacitors, current sources, and voltage sources, as shown in Figure 3(a). The chip layout is divided evenly into regular blocks. One grid node corresponds to a partition block. One current source connected to a grid node models the current drawn by the cells in the corresponding block. For simplicity, the current source waveforms are modeled as triangles, as shown in Figure 3(b). The current drawn by a current source is determined by a summation of currents drawn by those cells within its block. The average switching current of a cell is determined by its switching frequency and switching capacitance. In our experiments, the time between t_s to t_e (refer to Figure 3(b)) is set to 1ns. During other times, the currents flowing through a cell are very small. Our modeling of the power grid and current sources is similar to that of [16].

All decoupling capacitors in a block are lumped and represented by a capacitor connected to the grid node. The decoupling capacitance (decap) comes from two sources. The first are intentionally inserted decaps, and the second are background decaps from the standard cells. Standard cell decaps can be computed from cell types and sizes using information in a cell library. If a decap is inserted far away from a noisy area, it may not be helpful to ease the power noise (as explained in [16]). Decap efficiency-degradation effects will be considered naturally in the grid simulation. If cells in a block switch more frequently, the block current drawn will be larger, the block voltage drop will increase, and the block will require more decaps to reduce power noise. A simulator will determine how serious the voltage drop is for each block.

Power pads are connected to certain grid nodes. They are modeled as voltage sources. Using flip-chip packaging, pads can be inserted at internal grid nodes. Their locations are not limited to grid periphery; they can also be inserted in the interior of the chip. For simplicity, in our model we insert power pads uniformly on the grid. By performing transient analysis, we can calculate the voltage profile of each grid node.

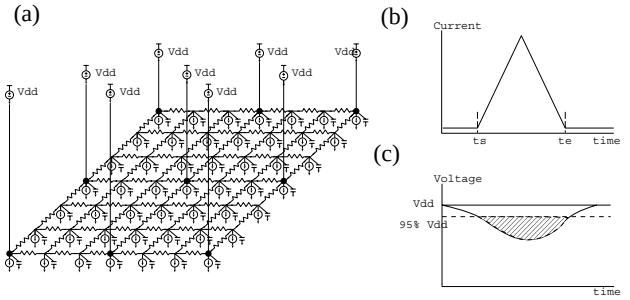


Figure 3. (a) Power grid model (b) current source waveform (c) Power voltage drop and excess noise area (ENA)

Because of the grid resistance, when a large current is drawn, a big voltage-drop may follow, as illustrated in Figure 3(c). Power grid voltage-drop affects chip performance. We assume

that a tolerable voltage drop threshold value is known. A voltage drop lower than the threshold is considered safe, not likely to cause timing violations or system malfunction. In our experiments, the voltage margin threshold is set at 5% from the ideal voltage. A typical noise margin can be set between 5~10%.

When voltage drop occurs in the power grid, the delay of cells connected to it changes. As described in [2], the pin-to-pin cell delay can be modeled as an inverse-linear function of supply voltage. The slope of the linear function can be characterized by simulation. This is the model we use. In our experiments, for all cells we set the growth rate of the delay with respect to voltage-drop to the same value.

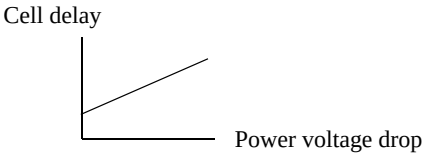


Figure 4. Cell delay model

We use three metrics to measure the power noise. The first metric is the deepest voltage drop on all grid nodes. This metric tells us the magnitude of the worst voltage drop on the chip. The second metric is the number of grid nodes that have a voltage drop greater than the threshold value. This metric reveals the overall power noise condition of the chip. We define the excess-noise-drop-area (ENA) for a node as the size of the area between the voltage margin threshold and the voltage drop. In Figure 3(c), ENA is the shaded area above the voltage drop curve. The third metric is the summation of ENA for all grid nodes. This third metric complements the second metric and gives us a better picture of the chip power noise. These three metrics quantify the local and global power noise. In the section on experiments, we will show the values of these three metrics.

2.2 Mutual-contraction-based wire-length prediction

Mutual contraction introduced in [7] is a metric to predict relative wire lengths before placement. A circuit is modeled by a graph with cells corresponding to nodes, and nets are represented as cliques with connections for each pair of nodes in a net. A weight is assigned to each connection. If a net k is connected with $d(k)$ nodes, then every connection c in this clique is assigned a weight given by (EQ 1). Other connection weighting methods have been discussed in [7], but (EQ 1) produces the best results.

$$w'(c) = \frac{2}{d(k) \times (d(k) - 1)} \qquad \text{(EQ 1)}$$

For a pair of nodes (u, v), $w'(u, v)$ is a weight of the connection between them. $\sum_x w'(u, x)$ denotes the sum of all weights on connections incident to u. A relative weight of a connection incident to u is defined as a ratio of the weight of this connection over the weight of all connections incident to u, as shown in (EQ 2).

$$w_r(u, v) = \frac{w'(u, v)}{\sum_x w'(u, x)} \qquad \text{(EQ 2)}$$

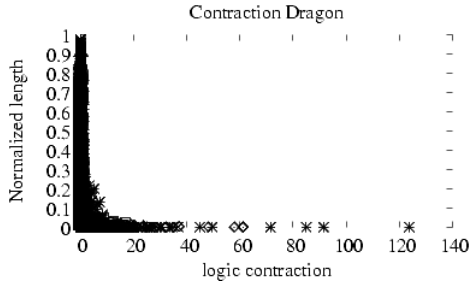
For a connection linking nodes x and y, the mutual contraction $MC(x, y)$ is computed using (EQ 3). This measure allows us to predict the relative wire lengths of connections.

$$MC(x, y) = w_r(x, y) \times w_r(y, x) \qquad \text{(EQ 3)}$$

Placers can be implemented using various methods and cost functions. Most placers try to minimize the total wire length. Mutual contraction is derived based on this assumption. In Figure 5, we show two graphs demonstrating the relationship between mutual contraction and distance among cells placed by two state-of-the-art academic placers, Dragon [18] and FengShui [1]. The results for six *MCNC* benchmarks (bigkey, frisc, s38584, clma and frisc) are combined and shown in Figure 5. First we compute the contraction value for each connection and then we perform placement. After placement, individual connection lengths are normalized by the chip dimension. In Figure 5, the x-axis measures the mutual contraction values, and the y-axis measures the normalized connection lengths. All benchmarks follow the same trend.

Both placers produce results in which the cell-pairs with larger mutual contraction tend to be closer. For the cell-pairs with smaller contraction, the variation of their distances is quite large.

(a)



(b)

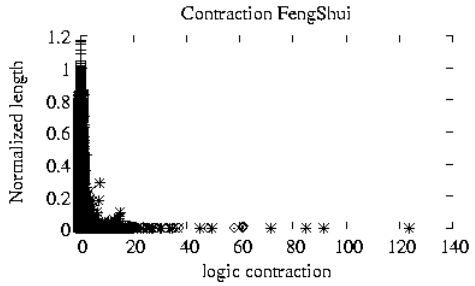


Figure 5. Contraction vs. placement wire length for benchmarks (a) Dragon (b) FengShui

From the placement results, we extracted the wire lengths and evaluated the correlation between the wire length and the contraction strengths. We refer to the nets with the top 30% highest contraction values as strong connections (Strong_Co). We compared the average wire length for strong connection nets (Strong_Co) to the overall average wire lengths (All_Co). For each benchmark we normalized these lengths with respect to its chip size (half perimeter). These results are shown in Table 1.

From the table we can see that the average strong connection length is only 4.85% of the chip dimension. However, the average connection length is about 38.06% of the chip's dimension. The last column shows the wire length standard deviation for strong connections. We can see that standard

deviation for strong connections are also very small. These results show that contraction can give a good prediction of the node neighborhood. Our extensive experiments suggest that as long as a placer minimizes the total wire length, the mutual contraction as a wire-length predictor is very effective.

Table 1. Wire length statistical results

	Strong_Co	All_Co	Strong_Co Deviation
bigkey	10.91%	38.47%	0.142
apex2	4.62%	35.53%	0.087
clma	2.01%	38.44%	0.083
s38584	2.86%	36.91%	0.081
frisc	3.19%	37.73%	0.117
ex1010	5.53%	41.28%	0.073
AVG	4.85%	38.06%	0.097

3. PREDICTION STEP: NEIGHBORHOOD-AWARE DECAP ALLOCATION

We decide decap allocation based on noise and timing weights for each cell. If we predict that a cell may experience excessive power noise, we assign a larger noise-weight to it and consequently we allocate more decap padding. We also estimate cell delay and interconnect delay. From the delay estimations and slacks, we compute cell timing criticality. If a cell has high timing criticality, we reduce its decap weight and decrease the allocated decap padding. Timing weights help us enforce timing constraints for the circuit.

3.1 Noise weights

The likelihood that a cell might have a large amount of power noise is estimated by its average current and by the currents of its neighbors. Neighborhood prediction is important, because even if a cell consumes much power, but most of its neighbors are quiet, this cell is not likely to suffer from extensive power noise. The neighborhood is defined in terms of layout distance. In this case, the neighborhood cells act as decaps. Using the pre-layout wire length estimates discussed in the previous section, we can predict the neighborhood of each cell.

Cell current consumption (CC) is a function of the cell's switching frequency and switching capacitance. Cell switching frequency can be estimated by feeding the circuit with input vectors and performing functional simulation. Another way to calculate the switching frequency is to calculate the switching probability. For a quick analysis, in our experiments, we use a probabilistic method as suggested in [17].

Cell-switching capacitance consists of a cell's intrinsic capacitance, input capacitance of fanout nodes, and wire capacitance. The intrinsic and input capacitances can be obtained from the netlist. Wire capacitance is unknown before placement, so we use a simple statistical wire-load model to predict it. The average lengths for nets of various degrees can be extracted from previous placements of similar designs. In our case, wire length statistics are averaged over all our benchmark circuits.

In the (EQ 4)-(EQ 5) we use the following notation: switch_freq(n) denotes the cell n's switching frequency, wcap(n) is the wire loading capacitance for n, fanout_cap(n) is the total input capacitance of n's fan-out cells, jcap(n) is n's junction capacitance. load_cap(n) is the total loading capacitance of n. CC(n) is n's current consumption. The expressions for computing cell current consumption (CC) are shown in (EQ 4) and (EQ 5).

$$\text{load\_cap}(n) = \text{wcap}(n) + \text{fanout\_cap}(n) + \text{jcap}(n) \quad \text{(EQ 4)}$$

$$\text{CC}(n) = \text{switch\_freq}(n) \times \text{load\_cap}(n) \quad \text{(EQ 5)}$$

Recall, that the connections whose mutual contraction values are among the top 30% are classified as strong connections that are expected to be short after placement. The cells connected by strong connections are expected to be in close proximity after placement. Those connections not classified as strong are deleted from the circuit graph and thus have no impact on neighborhood current-consumption computation. We define the 0-th level neighbors of a cell n is the cell itself. The $(i+1)$-th level neighbors of n include n's i-th level neighbors and all the nodes linked by strong connections to its i-th level neighbors.

We measure the neighborhood current consumption by computing the neighborhood-CCs (NCC). If a cell has a high NCC, we predict its power noise to be more serious. The neighborhoods and NCCs are defined for various levels. When using a higher level neighborhood, the neighborhood size will increase, so more neighbors of n will be involved when computing n's NCC. The 0-th level NCC of n is its CC. Computing cell n's i-th level NCC involves the lower-CC cells in n's i-th level neighbors. The NCC function is designed such that to compute consecutive levels of NCCs, a cell needs to remember only its 1-st level neighbors. This helps us save computation time and memory. The reason that we only consider those lower-CC cells in n's neighbors during the computation is that those cells may act as decaps and will bring down the current consumption in that area. The NCC of cells in a low-noise neighborhood will quickly decrease; however NCC for cells in noisy area will decrease less rapidly. This helps us filter out the noisy areas.

The i-th level NCC of a node n depends on the switching of its i-th level neighbors. In the first iteration, we compute the first level NCC of every node from the initial cell CC values. Based on the first level cell NCC results, we compute the second level NCC of every node. Higher level NCC can be computed following this iteration. Let $B(n)$ denote the 1-st level neighbors of n. $NCC(n)^i$ is the i-th level NCC of n. $A(n)^{i+1}$ is the set of nodes that are in $B(n)$ and have i-th level NCC not larger than $NCC(n)^i$. The $(i+1)$-th level NCC of n is the average of the i-th level NCC of nodes in $A(n)^{i+1}$. The expression for computing $NCC(n)^{i+1}$ is defined by (EQ 6) and (EQ 7).

$$NCC(n)^{i+1} = \frac{1}{\left|A(n)^{i+1}\right|} \times \sum_{k \in A(n)^{i+1}} NCC(k)^i \quad \text{(EQ 6)}$$

$$A(n)^{i+1} = \{NCC(k)^i \le NCC(n)^i, \forall k \in B(n)\} \quad \text{(EQ 7)}$$

For example, in Figure 6(a), we show a small netlist. The edges drawn are all strong connections. The number beside each node is the CC. The cells involved in n's second level NCC computation are shown in Figure 6(b). *L_i* stands for the

i-th level NCC. The numbers beside and below each node are their level NCCs. For the *0*-th level NCC, only those cell NCCs no bigger than the target cell NCC are shown. For instance, we compute $NCC(n)^1$ by averaging the *0*-th level NCC of *{a, b, n}*, because $NCC(a)^0$, $NCC(b)^0$ are all smaller than $NCC(n)^0$. However, to compute $NCC(b)^1$, only $NCC(a)^0$ and $NCC(b)^0$ are averaged, because among *b*'s *1*-st level neighbors, only $NCC(a)^0$ is smaller than $NCC(b)^0$. The $NCC(n)^2$ is computed by averaging $NCC(a)^1$, $NCC(b)^1$ and $NCC(n)^1$, because $NCC(a)^1$ and $NCC(b)^1$ are all smaller than $NCC(n)^1$. As the NCC level increases, and more low-CC cells are involved in the computation, the cell's NCC decreases. If a cell has many low-CC neighbors, its NCC decreases rapidly. However, from this example, we can also see that although higher-level NCC could involve many cells, those cells in the lower level neighbors still play a significant role in the NCC computation.

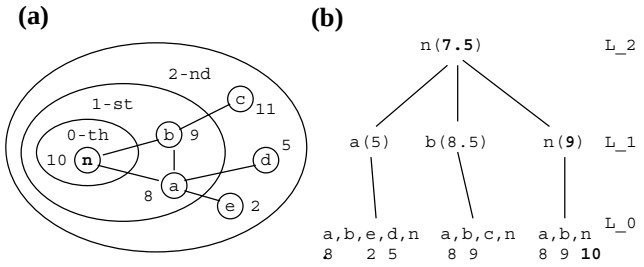


Figure 6. (a) Computing NCC of u (b) Computing $NCC(n)^2$

The purpose of the NCC computation is to determine the noisy areas. The high-CC cells with few switching neighbors will be filtered out. Only the clustered high-CC cells will retain their high NCC.

The noise weight for a cell is computed from the normalized cell NCC. *le* is the neighborhood level to compute NCC. Suppose that the max_NCC^{le} is the maximum *le* -th level NCC over all the cells. The normalized cell NCC for *n* is computed by dividing $NCC(n)^{le}$ by max_NCC^{le}. $nw(n)$ is the noise weight for the cell *n*. The noise weight function of a cell is shown in (EQ 8). In Section 5, we experiment using different *le* 's and observe their impact on the distribution of noise weighting. We find that setting *le* =4 leads to best noise weight distribution.

$$nw(n) = \left(\frac{NCC(n)^{le}}{\text{max\_NCC}^{le}}\right) \quad \text{(EQ 8)}$$

3.2 Timing weights

Besides considering the power noise factor, we also need to account for the timing factor. If cells are timing-critical, we do not add large decaps to them. Adding a large decap padding area to cells on a critical path may increase distances between the cells and consequently increase the interconnect delay. The criticality of a cell is computed using its slack. $slack(n)$ denotes the slack of a node *n*. Slack for each cell can be computed from its input signal arrival and required times. *max_slack* is the maximum slack of all the nodes. $slack(n)$ is normalized by the max_slack . *tw_exp* is the timing weight exponent. $crit(n)$ is the criticality of a node. $tw(n)$ is the timing weight of a node *n*. If a node has a smaller slack, it is more timing-critical and its $crit(n)$ will be higher. The timing criticality of a node is computed from (EQ 9). With bigger tw_exp , the criticality difference between the highly critical and non-critical cells will be larger. The same criticality function has been used in [12]. Based on their experiments, we set tw_exp to 4.

$$crit(n) = \left(1 - \frac{slack(n)}{\text{max\_slack}}\right)^{\text{tw\_exp}} \quad \text{(EQ 9)}$$

The timing weight function is shown in (EQ 10).

$$tw(n) = 1 - crit(n) \quad \text{(EQ 10)}$$

3.3 Decaps allocation

The decap area weight function is a summation of the noise and timing weights. *decap_weight(n)* is the decap weight for a node *n*. *T_S* is the timing cost scale. $tw(n)$ and $nw(n)$ are all normalized to a value in the range $0 \sim 1$. Setting T_S higher will increase the timing weight and assign less decap to a timing-critical area. Decap weight function is shown in (EQ 11). In the experimental results of Section 5, we will evaluate the impact of setting T_S to different values.

$$\text{decap\_weight(n)} = nw(n) + \text{T\_S} \times tw(n) \quad \text{(EQ 11)}$$

We allocate decap area according to the node's decap weights. Since we use the standard cell flow, the cell height and decap height are both fixed. The total decap width is computed by multiplying the total cell width by a decap ratio. Let *TCW* denote the total cell width and *DR* denote the decap ratio: the total decap width is $TCA \times DR$. A default value for *DR* is 0.2. Bigger *DR* may reduce power noise more, but at a cost of increased chip area, increased power consumption, or degraded chip timing. The portion of decap allocated to a cell *n* will be the ratio of decap weight of *n* and the summation of decap weights for all nodes. d_width(n) is the decap width of node *n*. The decap weight function is shown in (EQ 12).

$$\text{d\_width(n)} = \frac{\text{decap\_weight(n)}}{\sum_{k \in N} \text{decap\_weight(k)}} \times TCW \times DR \quad \text{(EQ 12)}$$

3.4 Experiments

In this subsection, we demonstrate the results of our neighborhood-aware decap allocation algorithm. We use the benchmark circuit ex1010 in this demonstration. The quantitative results for all benchmarks will be shown in Section 5.

First we perform the neighborhood prediction and compute various level-NCCs. Then we do placement using Dragon. In Figure 7 we show the cell NCC distribution for various NCC levels. In this example, we set T_S =0, showing only the effect of power noise. In Figure 7(a), we show the top 55% current-consuming cells with neighborhood level *0*. Neighborhood level *0* means that in computing a cell's NCC, only the current consumed by this cell is accounted for. We observe that in Figure 7(a) the upper-left and lower-left areas are very dense and could be power-noisy. Other areas also have numerous highly switching cells. In Figure 7(b), we show the cell NCC distribution considering their first-level neighborhoods. We use the minimum NCC of those cells shown in Figure 7(a) as the threshold value, showing in Figure 7(b)(c)(d) only those nodes with NCC greater than the threshold value. From (b), we can see that the cells in the right and center areas become more sparse, which means that number of high-NCC cells decreases in those areas. However, the power-noisy areas in the upper-left/lower-left corners are still dense and become more visible. Figure 7(c) and (d) show the results with neighborhood levels 2 and 4. As the NCC level increases, the sparse area becomes

even sparser. The number of high-NCC cells keeps decreasing in those areas. This experiment shows that the iterative NCC computation scheme is effective for isolating the noisy areas. The high-CC cells with low-CC neighbors are filtered out. More decaps can be allocated to those expected noisy areas to reduce power noise.

The power-grid simulation result is shown in Figure 8 for the case where the NCC level is equal to 4. The power voltage is 1.8V. The grid granularity is 20x20. The unit in the x and y dimension is mm. There are several power pads in the middle and on the periphery of the grid. We assume the chip switching frequency is 100MHz. Grid node decaps and current profiles are determined as described in Section 2.1. The worst grid voltage drop is recorded for each grid node. Figure 8(a) shows the result when decaps are distributed uniformly for all cells, and our weighting technique has not been applied. Figure 8(b) shows the result of decaps distributed according to our prediction-based weighting method. We use the same cell placement for (a) and (b), so it is easier to compare the difference in power noise. We observe that, in both figures, the biggest voltage-drop occurs at the upper-left and lower-left parts of the chip, which is just as predicted in Figure 7(d). The lowest grid node voltage is 1.66V in Figure 8(a) and 1.72V in (b). The results show that our prediction-based decap weighting method is effective in reducing the power noise. Figure 8(b) uses the same placement as (a), so this placement contains overlaps. Figure 8(c) shows the result after running a new placement. This placement is legalized and the power noise reduction is similar to (b). In this subsection, we show only part of the experimental results, more results will be shown in Section 5.

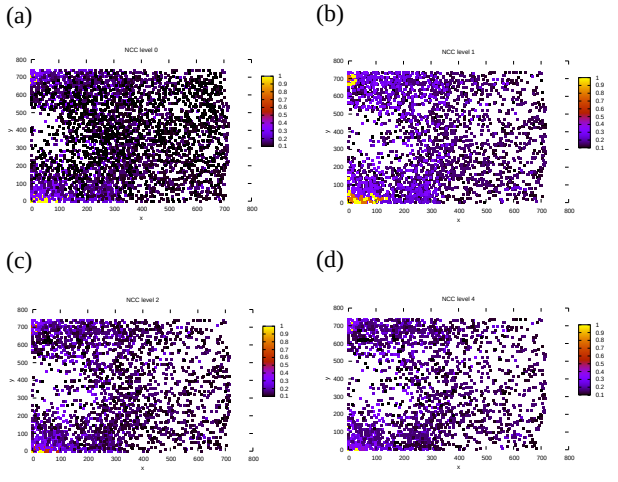


Figure 7. Neighborhood-aware power noise prediction for Dragon (a) cell current consumption with neighborhood level 0 (b) cell current consumption considering the 1st level neighborhood (c) 2nd level neighborhood (d) 4th level neighborhood

4. CORRECTING STEP: GATE-SIZING FOR POWER NOISE AND TIMING

After assigning decap padding to cells, we carry out placement and power grid analysis. There are several placers which attempt to spread out highly-switching cells across the chip [4][13]. In our experiments we use the publicly available

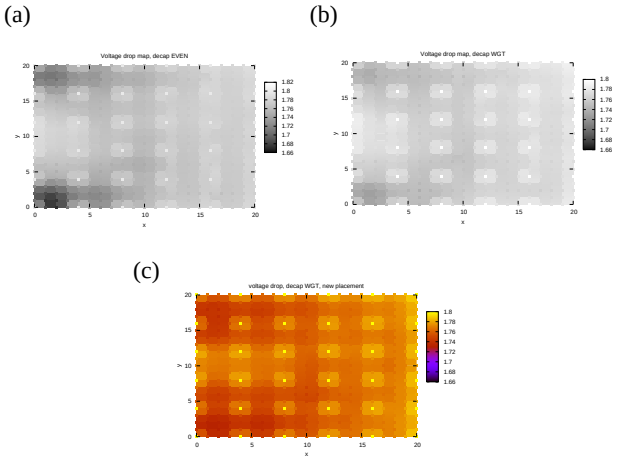


Figure 8. Power grid simulation with Dragon placement (a) Decap uniformly distributed (b) Decap distribution using prediction weighting (c) Decap distribution using predictive weighting and new placement

academic placer Dragon [18], which does not have the capability of spreading the frequently switching cells. After placement, long interconnect delays can be reduced by buffering or gate-sizing to meet timing constraints and further reduce power noise. In this section, we describe gate-sizing algorithms to optimize power noise and timing. The first algorithm is based on a Sequence-of-Linear-Programs (SLP), and the second algorithm uses a budgeting-based heuristic. Both gate-sizing algorithms take into account power-noise optimization.

4.1 Correcting step: a SLP optimization

The first algorithm uses an SLP technique, which solves a linear program (LP) in each iteration. In each iteration the coefficients of the LP are updated and a new LP is derived for the next iteration. In each linear program formulation, three types of constraints are considered: timing, area, and power noise. The objective of each linear program is to minimize the total power consumption and to reduce the power noise. We will discuss each type of constraint separately.

4.1.1 Timing constraints

The circuit is modeled as a graph, G. Nodes in the graph correspond to the cells, and edges represent the source-sink relationships in the circuit. Note that the graph model employed here uses edges rather than connections as in Section 2.2.

We model cell delay using a gain-based model. g_u is the fan-in arrival time of u. d_u is the node delay of u. The timing constraints are stated in (EQ 13):

$$g_u + d_u \le g_v, \ \forall e(u,v). \qquad \text{(EQ 13)}$$

When calculating node delays, we include the IR-drop effect on delay and loading capacitance. V^{chip} is the ideal power supply voltage. V_u^{pn} is the actual supply voltage after power grid simulation. M_u is the intrinsic cell delay. L_u is the delay slope per unit of loading capacitance. S_u is the gate size of a cell u. $FO(u)$ is the set of fan-out nodes for a cell u. μ_w is the size-1 input capacitance for a cell w. $W(u)$ is the wire

capacitance loading for cell u. The node delay function is computed according to (EQ 14). The part of equation in the bracket is contributed by the traditional gain-based delay model (EQ 21). (V^{chip}/V_u^{pn}) is the delay scaling from supply voltage. In general L_u is not constant, but the load dependence of the delay can be assumed linear in the neighborhood of the size S_u.

$$d_u = \frac{V^{chip}}{V_u^{pn}} \times \left[M_u + \frac{L_u}{S_u} \times \left(\sum_{w \in FO(u)} (\mu_w \times S_w) + W(u) \right) \right] \quad \text{(EQ 14)}$$

The non-linear timing constraints like those in (EQ 14) cannot be used directly in a linear programming formulation. We apply the first order Taylor's expansion to transform (EQ 14) into a linear equation.

We calculate the derivatives of a node delay with respect to the gate size for all the fan-out nodes and the node itself. d_u^0 denotes the node delay when gate size vector S equals to s^0. $w \in (u, FO(u))$ denotes the node u and its fan-out nodes. $((\partial d_u)/(\partial S_w))^{s = s^0}$ is the derivative of d_u function with respect to S_w for the gate size vector s^0. $(S_w - S_w^0)$ is the difference between the new gate size and the current gate size. Using Taylor's expansion, (EQ 14) is transformed to (EQ 15).

$$d_u = d_u^0 + \sum_{w \in (u, FO(u))} \left(\frac{\partial d_u}{\partial S_w} \right)^{s = s^0} \times (S_w - S_w^0) \quad \text{(EQ 15)}$$

d_u^0, $((\partial d_u)/(\partial S_u))$ and $((\partial d_u)/(\partial S_w))$ can be computed using (EQ 16), (EQ 17) and (EQ 18).

$$d_u^0 = \frac{V^{chip}}{V_u^{pn}} \times \left(I_u + \frac{L_u}{S_u^0} \times \left(\sum_{w \in FO(u)} (\mu_w \times S_w^0) + W(u) \right) \right) \quad \text{(EQ 16)}$$

$$\frac{\partial d_u}{\partial S_u} = \frac{V^{chip}}{V_u^{pn}} \times \frac{L_u}{-(S_u)^2} \times \left(\sum_{w \in FO(u)} (\mu_w \times S_w^0) + W(u) \right) \quad \text{(EQ 17)}$$

$$\frac{\partial d_u}{\partial S_w} = \frac{V^{chip}}{V_u^{pn}} \times \frac{L_u}{S_u^0} \times \mu_w, w \in FO(u) \quad \text{(EQ 18)}$$

Since the linear approximation of (EQ 14) by (EQ 15) is effective only for the gate sizes close to the initial values, we add the gate size change boundary constraints. S_i^L and S_i^U are the lower and upper bounds for the new gate sizes. GS_SCALE is the gate scale limit allowed in each iteration. The gate size boundary constraint is stated in (EQ 19). The upper and lower gate size bound can be calculated using (EQ 20). If we select a GS_SCALE too small, the number of SLP iterations will be large before the optimization converges. If we select a GS_SCALE too large, the convergence will be very difficult. We perform several experiments to select the scaling value that can lead to efficient convergence. We use GS_SCALE =1.2 as default. In a typical standard cell library, most of the gates are available in sizes between 1 and 4 (inverters are in sizes between 1 and 8). Gate determined by the sizing algorithm should be in the range provided by the cell library.

$$S_i^L \le S_i \le S_i^U, \ \forall i \in N \quad \text{(EQ 19)}$$

$$S_i^L = S_i^0 / \text{GS\_SCALE}, \quad S_i^U = S_i^0 \times \text{GS\_SCALE} \quad \text{(EQ 20)}$$

4.1.2 Area constraints

In the gate-sizing optimization, we add constraints to guarantee that the summation of the gate and decap areas does not change after the optimization, so that the chip area remains the same. We try to avoid a large decap re-allocation. Large decap area re-allocation might cause displacement of a large number of cells, which in turn could affect design convergence. Our idea is to divide the chip area into several equal-sized blocks. The summation of gate area and decap area in each block stays the same during the sizing optimization. B is the set of all blocks. β_u is the cell area increase ratio when its size increases by u. C_u is the decap padding area for the cell u. K_i is the summation of the cell and decap areas in a block after the first placement. The area constraints are stated in (EQ 21).

$$\sum_{u \in Block(i)} (\beta_u \times S_u + C_u) = K_i, \ i \in B \quad \text{(EQ 21)}$$

4.1.3 Power noise constraints

The effectiveness of a decap to reduce power noise depends on its size and distance from the power-noisy area. We need a sufficient amount of decap in the power-noisy area to reduce the noise. To handle the power noise constraints, we divide the chip area into several equal-sized blocks. The power-noise constraints guarantee that the summation of decaps in a block is greater than the summation of switch currents of all the gates in the block multiplied by a scalar value for power noise improvement.

Suppose that $\gamma_u \times S_u$ is the average current drawn by a cell u. γ_u can be computed using the cell switching frequency and loading capacitance. Z_m is the largest ratio of block decap over block current consumption among all the blocks in the current solution. PN_IMP is an improvement factor for power noise. Z is the lower bound ratio between the block current drawn and decap in the optimization. Z is computed as a product of Z_m and PN_IMP. The power noise constraint is stated in (EQ 22), and the formula for Z is shown in (EQ 23). We set the default value of PN_IMP to 1.2, which means the expected improvement of block decap over block current consumption is 20%. PN_IMP can be set higher for more improvement.

$$Z \times \sum_{u \in Block(i)} (\gamma_u \times S_u) \le \sum_{u \in Block(i)} C_u, \ i \in B \quad \text{(EQ 22)}$$

$$Z = Z_m \times \text{PN\_IMP} \quad \text{(EQ 23)}$$

4.1.4 The gate-sizing formulation

Constraints for the gate-sizing formulation include timing, power noise, and area. The optimization objective consists of two parts: the total power consumption and the weighted total decap area summation. $pc(S_u)$ is the power consumption for a cell u. $(V^{chip} - V_u^{pn})$ is the voltage-drop experienced by a cell u. Those cells whose voltage V_u^{pn} differs more from V^{chip} will be assigned more decap. BAL is the balancing factor. BAL is computed using (EQ 24). BAL is a normalizing factor between the power and noise cost function, enabling them to be compared appropriately. $NCOF$ is the noise weighting in the objective function. Its default value is 5, because we put greater effort on optimizing power noise. When $NCOF$ increases, more optimization effort will be put on reducing power noise.

$$\left(\sum_{u \in N} pc(S_u) \right) / \left(\sum_{u \in N} (V^{chip} - V_u^{pn})^2 \times C_u \right) = BAL \quad \text{(EQ 24)}$$

The gate-sizing objective function is shown in (EQ 25). The complete gate-sizing formulation is as follows:

Gate-sizing optimization for timing and power noise:

$$\underset{\text{Min}}{\sum_{u \in N}} [pc(S_u) - NCOF \times BAL \times (V^{chip} - V_u^{pn})^2 \times C_u] \quad \text{(EQ 25)}$$

Subject to:

$$g_u + d_u \le g_v, \ \forall e(u,v) \quad \text{(EQ 26)}$$

$$d_u = d_u^0 + \sum_{w \in (u, FO(u))} \left(\frac{\partial d_u}{\partial S_w}\right)^{s=s^0} \times (S_w - S_w^0) \quad \text{(EQ 27)}$$

$$\sum_{u \in Block(i)} (\beta_u \times S_u + C_u) = K_i, \ i \in B \quad \text{(EQ 28)}$$

$$Z \times \sum_{u \in Block(i)} (\gamma_u \times S_u) \le \sum_{u \in Block(i)} C_u, \ i \in B \quad \text{(EQ 29)}$$

(EQ 26) and (EQ 27) capture the timing constraints. (EQ 28) states the area constraints, and (EQ 29) expresses the power noise constraints.

After setting up the initial linear programming (LP) formulation and solving it, we obtain a new gate-size configuration that can improve the LP objective function. Using the new solution, we update the coefficients of the linear equations and solve the LP problem again. We can continue this iteration until the optimization converges. The SLP iteration is stopped when improvement becomes insignificant. In our implementation, if the total decap area increment in the current iteration is less than 10% of the decap area increment in the previous iteration, the SLP optimization is stopped. In the experiments, we will evaluate the improvement gained when applying different numbers of iterations.

4.2 Correcting step: budgeting-based heuristic

The SLP-based gate sizing algorithm can produce very high-quality results if we continue the iteration. Although SLP is efficient, the run time might still be too high for big circuits. In this section, we propose a heuristic gate-sizing algorithm that takes timing, power noise, and current consumption into account and that can achieve good results in a short time. In the following paragraphs, we discuss the case in which the critical-path timing constraint is larger than the current critical-path delay. In this case, we need only to down-size the gates. For the case in which the current critical-path delay exceeds the path-delay constraint, we can first uniformly increase the size of every gate until the path-delay constraint is satisfied. Next, our gate-sizing heuristic can be applied to reduce the gate sizes.

The gate-sizing heuristic is based on an iterative scheme. In each iteration, we resize gates a little according to weights assigned to them. We first compute the timing, power noise, and current consumption weight for each cell. Power noise weight $pn(n)$ is computed using (EQ 30). $ic(n)$ is the current consumption of *n*. Timing weight $tw(n)$ is from (EQ 10). The sizing weight, sizing_wgt(n) , is shown in (EQ 31).

$$pn(n) = (V^{chip} - V_n^{pn})^2 \quad \text{(EQ 30)}$$

$$\text{sizing\_wgt(n)} = pn(n) + ic(n) + tw(n) \quad \text{(EQ 31)}$$

We define a cell's gate level as the maximum level of gates for all paths from primary inputs or FFs to this cell. To guarantee that the resized gates will not cause timing violations, we resize gates level-by-level following the reverse gate-level order. As we resize the gates, the new cell-required-time will be updated. We make sure that the increase of delay is less than the cell's original slack. For example, as shown in Figure 9, cell *a* is at a gate-level (*i-1*) and cells b,c are at the level *i*. The original arrival and required times for *a* are 5 and 6, respectively. The original slack of *a* is 1. If we reduce the size of *a*, its delay will increase whereas its required time will decrease. The maximum delay increase for *a* will be equal to its slack, which is 1. In our program, we define the amount of cell delay increment budget, $rbgt(n)$, as the minimum of the cell slack and cell sizing-weight multiplied by a cell-delay-increment-unit, INU .

$$rbgt(n) = min(slack((n), \text{sizing\_wgt(n)} \times INU)) \quad \text{(EQ 32)}$$

If we assign INU to a large value, cell-required-time will decrease quickly in the first few reverse-levels, and only cells in those levels will be resized. However, if we assign INU to a value too small, we will need many resizing iterations to finish the optimization. From our experiments, we observe that setting $INU = 0.1ns$ (which is a value of about the same order as the cell's intrinsic delay) can strike a good balance between the run time and quality.

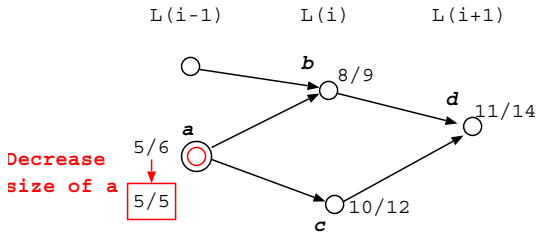


Figure 9. Required time update and gate sizing

After the node sizing-weight is computed, we update the cell delay and arrival times. Then we check to see if there is room for gate-sizing optimization. This is done by noting whether the reduction of a total slack in this iteration is greater than 10% of the slack reduction of the previous iteration. If the criterion for improvement is satisfied, we will continue the optimization; otherwise the algorithm stops.

After the optimization, many cells may have smaller sizes. We increase and relocate decaps in each partition area according to the updated current consumption, $ic(n)$. The partitions are as described in Section 4.1.2 and Section 4.1.3. The reason for relocating decaps only within a partition is to reduce the circuit performance disturbance.

The flow of the heuristic gate-sizing algorithm is shown in Figure 10.

5. EXPERIMENTS

We conduct our experiments using 0.18um technology. Several middle- and large-size benchmark circuits are selected from the MCNC benchmark suite. Columns 1 and 2 in Table 2 show the circuit information. Benchmark circuits have sizes ranging from 4199 to 23362 cells. The 3rd and 4th columns in Table 2 show the number of grid nodes and power pads for each circuit, respectively. TCW denotes the summation of all cell widths. For each benchmark, the available total decap width is

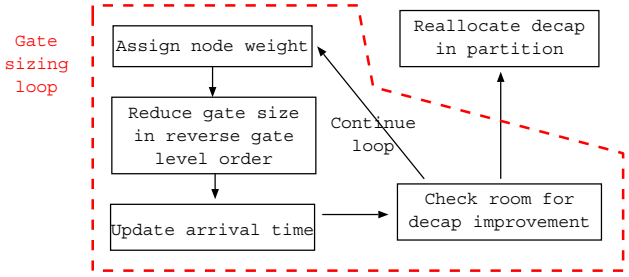


Figure 10. Heuristic gate sizing flow

given as a percent of the total cell width. We will experiment with varying total decap percentages. Since we assume a standard cell design style, the heights of the cells and decaps are the same. The sum of the decap and cell areas defines the total chip area. Circuits are placed using the fixed-die mode in Dragon. The default chip voltage is 1.8V and the voltage margin threshold is 5% of the ideal voltage. The experiments are run on a Linux Intel 2.4GHz machine.

Table 2. Benchmark information & wire length prediction

	$\|N\|$	#GN	#PN	TCW (um)
bigkey	4199	169	16	6983.6
apex2	4675	169	16	6956.9
clma	23362	1600	196	33515.0
s38584	13080	441	36	15100.4
frisc	7851	289	25	10665.9
ex1010	5838	289	9	11050.4

Figure 11 shows the experimental flow. We first run SIS [20] technology mapper with optimization objective for timing performance. SIS also does gate-sizing during synthesis. Based on the netlist characteristics of the input circuits, we perform the decap allocation prediction using the algorithm discussed in Section 3. Afterwards we change the cell widths to include decap padding, and perform the placement. We do not need to modify the placer to take decap allocation into account. After placement, we update wire capacitance and gate delay, and then perform the power grid analysis. Next, we determine voltage drops for all cells, update cell delay according to the new grid voltage, and do timing analysis with the new node delays. These are the results, after the prediction step, which form the input for the gate sizing. After the sizing optimization, we perform the grid analysis again. Cell delays are also updated to reflect the new grid voltages and then we perform timing analysis. These are the results after power noise correction.

.

5.1 Prediction scheme evaluation

To evaluate the decap prediction methods, we conduct experiments applying various strategies. First, we allocate no decaps to cells (NOC). Second, we distribute evenly decaps to all cells (EVEN). Third, we perform the prediction-based

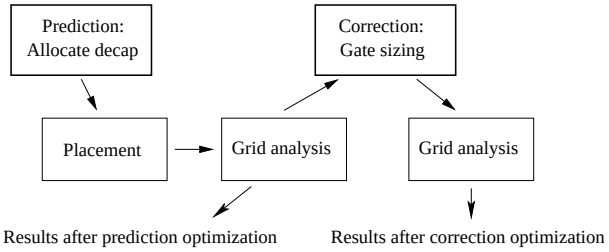


Figure 11. The experiment flow

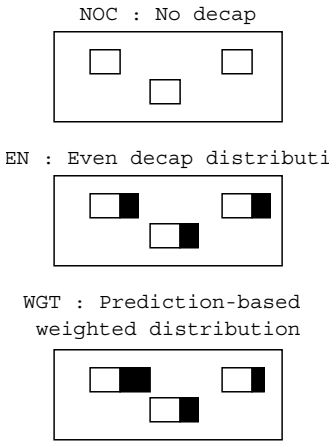


Figure 12. Illustration for different decap distribution (DD)

decap allocation (WGT) ignoring timing cost (T_S=0). Fourth, we perform prediction-based allocation including timing cost (T_S=1). When T_S=1, the decap allocation considers noise and timing weights as equally important. A graphic illustration of different strategies are shown in Figure 12.

The experimental results are shown in Table 3. T_S is the timing cost scale in (EQ 11). DD denotes various methods of decap distribution. IRD is the voltage-drop. SENA denotes the summation of excess noise area for all grid nodes in units of $Volt \times 10ps$. vioC is the number of grid nodes that have voltage drop greater than the voltage margin threshold. CritP is the critical path delay. IRD, SENA and vioC are all computed from the actual current waveform profiles. The last four rows show the normalized average results for all 6 circuits. The results are normalized with respect to the second strategy (EVEN).

From the average results in Table 3, we can see that for T_S = 0, the power noise, timing and total slack results are all improved when DD changes from NOC to EVEN and to WGT. Comparing the cases of EVEN and WGT, the IRD (IR-drop) decreases 27%, SENA (summation of excess-noise-area) decreases 51%, and vioC (grid node noise violence count) decreases 28%. This shows that our prediction-based decap allocation method is effective, and decaps are useful in reducing power noise. The timing also improves because voltage-drop decreases and node delays become shorter. When we increase timing weights and change T_S from 0 to 1 using prediction-weighting (WGT), timing results improve by 4%; however, power noise results become worse. The timing improvement is only minor when increasing the timing scale T_S.

Table 3. Experimental results after prediction, DR=0.2, different timing scale (T_S) and delay distribution methods (DD)

	DD	IRD(V)	SENA	vioC	CritP(ns)
bigkey	NOC	0.23	1261	0.88	4.43
	EVEN	0.17	457	0.62	4.34
	T_S=0,WGT	0.14	339	0.65	4.28
	T_S=1,WGT	0.19	478	0.61	4.15
apex2	NOC	0.34	2954	0.86	5.61
	EVEN	0.26	1646	0.74	5.41
	T_S=0,WGT	0.21	1439	0.76	5.37
	T_S=1,WGT	0.27	1747	0.74	5.17
clma	NOC	0.28	137	0.16	10.56
	EVEN	0.25	108	0.12	10.52
	T_S=0,WGT	0.17	40.5	0.08	10.49
	T_S=1,WGT	0.22	106	0.12	10.50
s38584	NOC	0.37	2121	0.90	9.01
	EVEN	0.29	937	0.76	8.81
	T_S=0,WGT	0.23	789	0.80	8.85
	T_S=1,WGT	0.23	849	0.77	8.31
frisc	NOC	0.17	282.23	0.58	17.90
	EVEN	0.12	42.9	0.15	17.67
	T_S=0,WGT	0.10	5.32	0.09	17.65
	T_S=1,WGT	0.22	78.67	0.14	17.20
ex1010	NOC	0.16	57.67	0.09	6.30
	EVEN	0.13	16.5	0.03	6.26
	T_S=0,WGT	0.07	0	0	6.23
	T_S=1,WGT	0.15	38.31	0.04	5.85
AVG	NOC	1.27	3.02	1.89	1.01
	EVEN	1	1	1	1
	T_S=0,WGT	0.73	0.49	0.72	0.99
	T_S=1,WGT	1.14	1.36	1.04	0.96

Table 4 shows the total wire length comparison for four cases NOC, EVEN, (T_S=0,WGT) and (T_S=1,WGT). The last row shows the normalized wire lengths. For each benchmark the chip size is the same for all experiments. The wire length in NOC is smaller than in other experiments, because with decaps absent, cells can be placed closer. For the other three experiments, the total wire lengths results are similar.

5.2 Decap ratio effect evaluation

In the experiment reported in Table 3, we use decap ratio (DR) as 0.2 of the total cell area. It is interesting to observe how the decap ratio affects power noise. We conduct additional

Table 4. Total wire length (um)

	NOC	EVEN	T_S=0,WGT	T_S=1,WGT
bigkey	349090	365035	361499	371117
apex2	502633	540598	537984	569483
clma	2319435	2555250	2572195	2540234
s38584	604227	657091	648058	649950
frisc	380906	412766	426334	394830
ex1010	900959	986241	936920	1009425
Avg	0.93	1.01	1	1.01

experiments using DR=0.1 and 0.3. We obtain placement from experiments in Table 3, scaling the chip width accordingly to scale the decap area. The number of rows and columns in the power grid do not change. We experiment with the case T_S=0, and the decap allocation methods EVEN and WGT. The normalized average results from all benchmarks are shown in Table 5. Those results are normalized to the case T_S=0 and the EVEN decap distribution in Table 3. From the results, we can see that as the decap ratio increases, the power noise results improve, although the timing results degrade slightly. Comparing DR=0.3 and 0.1 at DD=WGT, the IRD reduces 23%, the SENA improves 91%, the vioC improves 65% and the timing degrades 2.8%.

Table 5. Average experimental results after prediction T_S = 0 for different decap ratios (DRs)

DR	DD	IRD(V)	SENA	vioC	CritP(ns)
0.1	EVEN	1.03	1.48	1.30	0.99
	WGT	0.86	0.98	1.09	0.98
0.3	EVEN	0.80	0.57	0.65	1.013
	WGT	0.60	0.26	0.55	1.008

5.3 NCC level effect evaluation

The power noise results depend also on the NCC-levels and how the neighbors of a cell are predicted. If too few NCC-levels are used, many decaps will be allocated to those cells having a high-CC but a small neighborhood current consumption. However, since such cells are unlikely to suffer from a power noise problem, they should not be allocated decaps. If its NCC levels are too large, a neighborhood will cover too much chip area and will lose its meaning. According to (EQ 6) and (EQ 7), NCC computation depends strongly on a cell's neighbors at a particular level. The effect of remote neighbors on a cell's NCC is small. When the NCC-level exceeds a certain value, far-away-neighbors will not have significant impact on a cell's NCC. In the first four rows of Table 6, we show the results when using varying NCC levels. The results for all the benchmarks are averaged and normalized with respect to the case of NCC level being equal to 4. We show the results for NCC levels 0, 1, 4, and 8. From the results, we can see that NCC level 4 gives the best results. When NCC

levels are too small or too great, the power noise results become worse.

As stated in Section 3.1, we consider those connections to be strong whose mutual contraction comes within the top 30%. We computed cell neighborhoods based on those strong connections. We also experimented with differently defined neighborhoods. For example, instead of using only the strong connections to determine neighborhoods, we used all the connections. As long as there was a connection between a pair of nodes, we considered them to be first-level neighbors. In this case, cell NCCs could be influenced by cells which have been placed far away. In Table 6, we refer to the case of using only strong connections as *Strong_Co*. The case for using all connections to find neighbors is referred to as *All_Co*. From the results, the NCC level-4 with neighborhood defined by strong connections gives much better results than the NCC level-4 and neighborhood defined by all connections. When using strong connections, the IRD improves 30%, the SENA improves 3.1 times, and the vioC improves 1.7 times. Using too few NCC-levels may not cover enough neighbors and may not capture the neighborhood effect on power noise. However using too many NCC-levels may cover too much area which in turn may increase the estimation errors. Therefore, choosing a good neighborhood size is important.

Table 6. Average experimental results after prediction T_S = 0 for different NCC levels and neighbor definitions

NBR	NCC level	IRD(V)	SENA	vioC	CritP
Strong_Co	8	0.992	1.084	1.012	0.999
	4	1.0	1.0	1.0	1.0
	1	1.015	1.034	1.116	1.001
	0	1.055	1.692	1.172	1.002
All_Co	4	1.299	3.094	1.716	1.001

5.4 Grid design effect evaluation

The power grid design also has a big impact on power noise. In this experiment, we show noise results for various granularity grid and pad designs. Gnode# denotes the number of grid mesh nodes. PAD# denotes the number of power pads. x1 refers to using the original design. x4 stands for increased node or pad count by 4 times. For this experiment, the total grid area is fixed. If we use twice the number of the vertical and horizontal grid lines, the number of grid nodes increases 4 times. The width of the power grid lines will be reduced by half, and their resistance will double. Table 7 shows the normalized results for all benchmarks with respect to the case in the first row. From the results, we can see that power noise is less dependent on the grid size than on the pad number. The improvement on SENA and vioC, when increasing the pad number, is significant.

5.5 Results after power noise correction

Table 8 shows the experimental results after power noise correction. The second column shows various types of optimization. *si0* denotes the experiment in which T_S = 0, DD = WGT for weighting prediction, and no SLP gate-sizing optimization after placement. *si2 and si4* denote the experiments in which we run 2 and 4 SLP gate sizing iterations after placement. *si** denotes the case in which we repeat SLP iterations until the stop criterion is met. *hur* denotes the results for a heuristic sizing algorithm.

Table 7. Average experimental results after prediction T_S = 0 for different power grid and pad granularity

Gnode#	PAD#	IRD (V)	SENA	vioC	CritP (ns)
x1	x1	1.0	1.0	1.0	1.0
x4	x1	1.057	2.387	1.611	1.005
x4	x4	0.66	0.002	0.02	0.978

TDW is the total decap width. TCC is the total chip current consumption. RT is the run time for the SLP and *hur* optimization. The last five rows are the normalized average results for all the circuits, with respect to the case *si0*.

We observed that as we conducted more SLP optimization iterations, the results showed improvements in power noise, timing, and chip power consumption. The decap width increased substantially. Note that the summation of the decap width and cell width remains fixed. The total cell width decreases by the same amount as the total decap width increases. Comparing *si0* with *si**, voltage-drop improves by 43%, SENA becomes almost 0, decap area increases 3.4 times, and current consumption improves by 43%. The results from the heuristic sizing algorithm are close to *si** and require much less run time. The heuristic iteration number ranges from 18 (for bigkey) to 71 (for clma). The average of *si** iteration number is 15.

Table 8. Experimental results after correction, different gate-sizing algorithms

		TDW	IRD (V)	SENA	vioC	TCC (uA)	RT(s)
bigkey	si0	6983	0.14	339	0.65	7659	
	si2	12211	0.11	2.97	0.15	6911	16
	si4	16526	0.09	0.08	0.02	6363	33
	si*	21813	0.06	0	0	4831	181
	hur	21349	0.06	0	0	5519	4
apex2	si0	6956	0.21	1439	0.76	9316	
	si2	14415	0.11	33.7	0.38	7368	154
	si4	20717	0.09	6.26	0.16	6336	313
	si*	27759	0.06	0.05	0.01	4650	1503
	hur	26047	0.06	0.38	0.01	5239	5
clma	si0	33515	0.17	40.5	0.08	35168	
	si2	68814	0.18	24.6	0.04	26635	2572
	si4	98250	0.16	11.3	0.02	23592	5332
	si*	127134	0.12	2.47	0.01	18247	22860
	hur	120047	0.13	3.86	0.01	19839	51

Table 8. Experimental results after correction, different gate-sizing algorithms

		TDW	IRD (V)	SENA	vioC	TCC (uA)	RT(s)
s38584	si0	15100	0.23	789	0.8	16635	
	si2	28465	0.15	49.5	0.27	14045	22
	si4	37772	0.12	7.27	0.09	12440	43
	si*	46976	0.08	0	0	8990	164
	hur	46531	0.09	0.09	0.01	10215	12
frisc	si0	10665	0.10	5.32	0.09	7329	
	si2	18543	0.09	0.06	0.02	6113	43
	si4	24323	0.07	0	0	5485	84
	si*	29345	0.04	0	0	3943	328
	hur	28936	0.04	0	0	4594	8
ex1010	si0	11050	0.07	0	0	4248	
	si2	22145	0.07	0	0	3849	379
	si4	29390	0.06	0	0	3584	742
	si*	42437	0.03	0	0	2780	3513
	hur	41135	0.06	0	0	3096	8
AVG	si0	1	1	1	1	1	
	si2	1.9	0.83	0.11	0.27	0.84	1
	si4	2.6	0.75	0.05	0.10	0.76	1.9
	si*	3.4	0.57	0.01	0.01	0.57	8.9
	hur	3.3	0.64	0.01	0.02	0.64	0.14

5.6 Voltage drop profile

Figure 13 shows the voltage profiles for different optimization schemes at a grid node for the benchmark bigkey. In this experiment T_S is set to 0, so the optimization targets only the power noise reduction. WGT-si2 denotes using prediction-based weighting (WGT) and two iterations of SLP. si0 denotes the case with no SLP optimization. We can see that the voltage drop decreases as we do more SLP optimization and use our prediction-based weighting scheme.

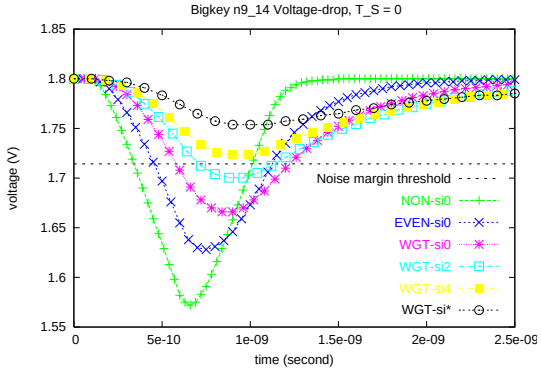


Figure 13. Power grid voltage-drop for bigkey

6. CONCLUSIONS

In this paper we addressed the power-noise problem considering timing constraints. Decap padding was added to each cell. We proposed a decap allocation flow which consists of *prediction* and *correction* steps. First we allocated decap to each cell based on predictions. Decaps were allocated to cells which were most likely to have large voltage drops. We also considered timing criticality in decap allocation so that the added decap area would not increase the critical path delay. A possible extension to improve timing is to include timing weights in the neighborhood decap computation. Currently our neighborhood-based decap allocation method only considers power weighting. For a cell in the critical path, its neighbors should have larger decap padding to reduce regional voltage drop. The delay for that critical cell can be decreased, if its voltage drop is reduced.

In the decap correction step, we performed gate sizing and re-allocated decaps based on a more complete information after placement and power grid analysis. For gate-sizing optimization, we proposed two algorithms. The first algorithm is based on the Sequence-of-Linear-Programs (SLP) method, and the second is a heuristic.

Both gate-sizing algorithms assume continuous sizing. However, in practice, it may not be possible. Our algorithms need to be adapted to discrete gate sizing. Our gate sizing algorithms also require interaction with grid simulation. After the gate sizing is done, we perform power grid simulation again and determine the new voltage drop profile. If necessary, gate sizing may be repeated. We think that the iteration scheme between gate-sizing and simulation is more practical than trying to solve both problems together, because of computational complexity.

Comparing the results achieved for uniformly assigned decaps against decaps assigned using our prediction-based weighting method, our method shows that the maximum voltage drop decreases by 27%, the sum of excess noise area decreases by 51%, and the grid voltage violations decrease by 28%. We found that changing the timing weight from 0 to 1 improved timing by 4%. The power noise results also showed improvement when the decap ratio was increased. Comparing decap ratio equals to 0.3 and 0.1 using prediction-based weighting (WGT), the IR-drop (IRD) reduces 23%, the total excess noise improves 91%, the grid violation count (vioC) improves 65%, and the timing degrades 2.8%.

Cell power noise was also affected by the extent of neighborhood levels that we used and how we defined neighbors. Our results showed that using contraction-predicted neighborhoods can produce 30% better results on IR-drop, rather than just using connections to estimate the neighborhoods. Different grid designs were shown to have impact on power noise. Power noise appears to be especially sensitive to power pad numbers.

For post-layout gate sizing, when using SLP, comparing *si0* with *si**, voltage-drop improves by 53%, total excess noise becomes almost 0, decap area increases 3.4 times, and current consumption improves by 53%. The gate-sizing heuristic algorithm produces results similar to si*; however the runtime is one-order of magnitude less. Even our biggest benchmark can be finished in 51 seconds.

These results show that our techniques are very effective and efficient for power noise reduction. For future more advanced design, power noise will become a more severe problem.

Although a good decap allocation scheme is an important part of reducing power noise, other steps like grid design, package design, and placement are also important. A more integrated approach is necessary to maintain power integrity. In advanced designs leakage power becomes a serious problem. Decaps contribute to leakage. In a case when leakage power is a limiting factor, our decap allocation method can be used to spread out highly switching cell clusters. Our method can be applied without decap insertion and can still improve power noise.

7. REFERENCES

[1] A. R. Agnihotri, S. Ono, and P. H. Madden, "Recursive Bisection Placement: Feng Shui 5.0 Implementation Details", *Proceedings of the 2005 International Symposium on Physical Design*, Pages 230 - 232, 2005.

[2] G. Bai, S. Bobba, and T. N. Hajj, "Static Timing Analysis Including Power Supply Noise Effect on Propagation Delay in VLSI Circuits", *Proceedings of Design Automation Conference*, Pages 295-300, June 2001.

[3] A. E. Caldwell, A. B. Kahng, S. Mantik, I.L Markow and A. Zelikovsky, "On wire length estimation for row-based placement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Pages 1265 - 1278, Sep 1999.

[4] G. Chen, S. Sapatnekar, "Partition-driven standard cell thermal placement", *Proceedings of the 2003 International Symposium on Physical Design*, Pages 75 - 80, 2003.

[5] H. H. Chen, "Minimizing chip-level simultaneous switching noise for high-performance microprocessor design", *IEEE International Symposium on Circuits and Systems*, ISCAS '96, vol.4, Pages 544 - 547, May 1996.

[6] H.-M. Chen, L.-D. Huang, I-Min Liu, M. Lai, and D.F. Wong, "Floorplanning with Power Supply Noise Avoidance," *Proc. of IEEE Asia and South Pacific Design Automation Conference*, Pages 427 - 430, January 2003 (ASPDAC-03).

[7] B. Hu, M. Marek-Sadowska, "Wire Length Prediction based Clustering and its Application in Placement", *Proceedings of Design Automation Conference*, Pages 800 - 805, June 2003.

[8] B. Hu, Y. Watanabe, A. Kondratyev, M. Marek-Sadowska, "Gain-based technology mapping for discrete-size cell libraries", *Proceedings of Design Automation Conference*, Pages 574 - 579, June 2003

[9] S. Hauck and G. Borriello, "An evaluation of bipartitioning techniques", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol 16, No. 8, Pages 849 - 866, 1997.

[10] T. Hamada, C.-K. Cheng, P. M. Chau, "A Wire Length Estimation Technique Utilizing Neighborhood Density Equations", *Proceedings of Design Automation Conference*, Pages 57 - 61, 1992.

[11] M. A. B. Jackson and E. S. Kuh, "Performance-driven placement of cell-based ic's", *Proceedings of Design Automation Conference*, Pages 370 - 375, 1989.

[12] A. Marquardt, V. Betz and J. Rose, ``Timing-Driven Placement for FPGAs," *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Pages 203 - 213, February 2000.

[13] B. Obermeier, F. M. Johannes, "Temperature-aware global placement", *Proceedings of the 2004 conference on Asia South Pacific design automation: electronic design and solution fair 2004*, January 27 - 30, 2004, Yokohama, Japan.

[14] M. D. Pant, P. Pant, D. S. Wills, "On-chip decoupling capacitor optimization using architectural level prediction", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 10, No. 3, Pages 319 - 326, June 2002.

[15] Z. Qi, H. Li, S.X.-D. Tan, L. Wu, Y. Cai, X. Hong, "Fast Decap Allocation Algorithm For Robust On-Chip Power Delivery", *Sixth International Symposium on Quality of Electronic Design* (ISQED), Pages 542 - 547, March 2005.

[16] H. Su, S. S. Sapatnekar and S. R. Nassif, "Optimal Decoupling Capacitor Sizing and Placement for Standard-Cell Layout Designs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol 22, No 4, Pages 428 - 436, April 2003.

[17] H.-C. Tsai, K.-T. Cheng, and V. Agrawal, "A Testability Metric for Path Delay Faults and Its Application," in *Proc. Asia and South Pacific Design Automation Conf.*, pp.593-598, Jan. 25-28, 2000.

[18] M. Wang, X. Yang and M. Sarrafzadeh, "Dragon2000: Standard-cell Placement Tool for Large Industry Circuits", *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Pages 260 - 263, November 2000.

[19] S. Zhao, K. Roy, and C.-K. Koh, ``Decoupling Capacitance Allocation and Its Application to Power Supply Noise Aware Floorplanning", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Special Issue on Physical Design)*, Vol 21, No 1, Pages 81 - 92, January 2002.

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