Multivoltage Floorplan Design

Qiang Ma and Evangeline F. Y. Young

***Abstract*—Energy efficiency has become a very important issue to be addressed in today's system-on-a-chip (SoC) designs. One way to lower power consumption is to reduce the supply voltage. Multisupply voltage (MSV) is thus introduced to provide flexibility in controlling the power and performance tradeoff. In region-based MSV, circuits are partitioned into "voltage islands" where each island occupies a contiguous physical space and operates at one voltage level. These tasks of island partitioning and voltage level assignment should be done simultaneously in the floorplanning process in order to take those important physical information into consideration. In this paper, we consider this core-based voltage island driven floorplanning problem including islands with power down mode, and propose a method to solve it. Given a candidate floorplan solution represented by a normalized Polish expression, we are able to obtain *optimal* voltage assignment and island partitioning (including islands with power down mode) *simultaneously* to minimize the total power consumption. Simulated annealing is used as the basic searching engine. By using this approach, we can achieve significant power saving (up to 50%) for all datasets, without any significant increase in area and wire length. We compared our approach with the most updated previous work on the same problem, and results show that our approach is much more efficient and is able to save more power in most cases. We have also studied two other approaches to solve the same problem, a simple dynamic programming approach and a lowest possible power consumption approach. Experimental results show that ours can perform the best among these three approaches. Our floorplanner can also be extended to minimize the number of level shifters, to address a *minVdd* version of the problem and to simplify the power routing step by placing islands close to their corresponding power pins.**

***Index Terms*—Floorplanning, low power, voltage island, voltage scaling.**

I. Introduction

THERE ARE two kinds of power consumption: dynamic and leakage. Dynamic power is caused by charging and discharging of the load capacitance during switching, while leakage power is due to subthreshold currents when a device is turned off. Energy efficiency is now an important issue in today's system-on-a-chip (SoC) designs because of the increasing power density and the popularity of portable systems. There are many techniques to reduce power consumption. One of the most effective ways is by lowering the voltage supply. Multivoltage design is thus introduced to provide "just enough" power to support different functional operations. Both dynamic and leakage power consumption can be reduced in multivoltage designs. For dynamic power, a minor adjustment to the voltage level can result in a significant reduction in power consumption, which is proportional to the square of the voltage. For leakage power, the consumption can be reduced by powering down some components when they are inactive.

Multivoltage designs involve partitioning of a chip into areas called "voltage islands" that can be operated at different voltage levels, or be turned off when idling. With the use of voltage islands, the chip design process is becoming more complicated. We need to solve the problems of island partitioning, voltage assignment, and floorplanning simultaneously under area, power, timing, and other physical constraints. These problems must be solved at the same time since their results will mutually affect each other. Also, there are other issues to be considered. For example, voltage islands should be placed close to their corresponding power pins in order to make power routing easier and to reduce IR drop. Besides, each island requires level shifters to communicate with others and overhead in area and delay will be resulted. These additional issues have created many new challenges in generating floorplans for multivoltage designs. An example is shown in Fig. 1. In this example, the possible voltage levels of each core and groupings of similar inactive periods (to generate islands with power down mode) are shown on the right hand side. Assuming that the number of islands is three, one possible partitioning is to group cores A, B, and C as one island operating at voltage 1.0-V, core D on its own as one island at voltage 1.5-V and cores I, K, L, and M as one island at 1.2-V. Notice that other cores will be operated at the chip-level voltage. The island containing I, K, L, and M can be powered down during sleep, as well as the island containing the single core D. A candidate floorplan solution for such a partitioning and voltage assignment is shown on the left.

There are several previous papers addressing this voltage island-driven floorplanning problem. One recent work is by Lee *et al.* [5]. Given a netlist without reconvergent fanouts, a voltage assignment (with two voltage levels of VDDL and VDDH) is first performed on the netlist according to the timing requirement before the floorplanning step. Level shifters are then inserted into the nets according to the voltage assignment

Manuscript received June 16, 2009; revised October 11, 2009. Current version published March 19, 2010. This work was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 4184/07). The preliminary version of this article appeared in the Proceedings of the 2007 International Conference on Computer-Aided Design [6]. This paper was recommended by Associate Editor, I. Bahar.

Q. Ma is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, IL 61821 USA (e-mail: qiangma1@illinois.edu).

E. F. Y. Young is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, China (e-mail: fyyoung@cse.cuhk.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2042895

result when a VDDL block drives a VDDH block. At the end, a power-network aware floorplanner is invoked to pack the blocks such that the power-network resource, estimated as the sum of the perimeters of the voltage islands, will be minimized. As a result, blocks in the same voltage island will be placed close to each other. In their approach, the voltage assignment step and the floorplanning step are done separately. Hu *et al.* [3] have also considered this simultaneous island partitioning, voltage assignment, and floorplanning problem in SoC designs. Simulated annealing is used as the basic searching engine. Given a candidate solution, perturbations are performed to split an island, change the voltage of an island or change all the islands of one voltage to another voltage. Chip-level floorplanning is then performed to find a floorplan in which compatible islands (islands with the same voltage) are likely to be adjacent. An island merging process is then applied to reduce the number of islands. At the end, island-level floorplanning is done to each newly formed island to shrink its area. The whole process is repeated until a satisfactory solution is obtained. Their approach does not consider islands with power down mode and the search space is large. Mak and Chen [7] have also addressed this problem on SoC designs. Given a floorplanning input, the voltage assignment and island partitioning problem is formulated as a 0–1 integer linear program. In their approach, a few candidate floorplan solutions are generated based on metrics like area and interconnect cost, then voltage assignment and partitioning are performed on these candidate floorplans using the integer linear programming approach to identify the best candidate solution. A fragmentation cost (number of adjacent cores operating at different voltages) is used to model the power network complexity but this cost is not related to the number of islands directly. There are some other previous works which address issues like reliability [12] and temperature reduction [4] in SoC voltage island partitioning and floorplanning. For island partitioning, Wu *et al.* [11] minimized the number of voltage islands after placement. Guo *et al.* [2] addressed the voltage assignment and voltage island generation problem in placement to minimize the number of level shifters. Cai *et al.* [1] proposed a voltage island generation flow in standard cell-based designs to reduce power consumption under performance constraint and to reduce layout overheads caused by cell clustering to form islands.

In this paper, we present a floorplanning method for SoC designs that is tightly integrated with island partitioning and voltage assignment. A preliminary version of it has appeared in [6]. Simulated annealing is used with normalized Polish expression (NPE) [10] as the floorplan representation.[1] Our cost function now considers area, wire length, power, and level shifter usage.[2] NPE is used because the slicing tree structure is a suitable data structure on which island partitioning and voltage assignment can be done *optimally and efficiently*. In each step of the annealing process, a candidate floorplan solution is generated on which *optimal* island partitioning and voltage assignment will be performed simultaneously to compute the smallest possible power consumption for that candidate floorplan solution. This is done by dynamic programming with an efficient cost table updating technique. In this way, we can integrate the three steps closely, and reduce the searching space (instead of doing voltage assignment by the "move" operations of the annealing process as in [3]). In this floorplanning framework, we can also generate islands with power down mode to optimize the total power consumption further. We compared our approach with the most updated previous work [7] on the same problem, and results show that our approach is much more efficient and is able to save more power in most cases, while with less area overhead. Similar to [7], timing is considered according to the assumption that the given voltage levels of each core are all feasible for achieving timing closure. To further study this problem, we have developed two other methods to solve this voltage island-driven floorplanning problem. One of them is a simplified version of the dynamic programming approach, and the other one assumes a lowest possible power consumption for each core. We compare these two methods with our approach and experimental results show that our approach can perform the best. The simplified dynamic programming approach performs a little bit inferior to our approach in terms of solution quality, with a 5.37% reduction in running time on average. The lowest possible power approach consumes less execution time and achieves maximum power saving, but the number of voltage islands generated is relatively large, while our approach can generate a solution of which the power consumption is fairly close to the lowest possible value, while the number of voltage islands is much smaller. In addition, we have also extended our floorplanner to consider the area and power overhead of level shifters, to solve a *minVdd* version of this voltage island driven floorplanning problem (in which each core, instead of having a set of discrete feasible voltage levels, has a minimum voltage level above which the core can be operated properly and achieve timing closure) and to consider the ease of power network routing (proximity to power pins and shapes of voltage islands). By using our approach, we can achieve significant power savings (up to 50%) for all datasets, without any significant increase in area and wire length.

We will define the problem in Section II, we will then discuss the methodology used in Section III. The experimental results will be reported in Section VI before the conclusion and discussion in the last section.

II. Problem Formulation

In this problem, we are given a set of n cores with areas $A_1, A_2, \ldots, A_n$ and aspect ratio bounds $[l_i, u_i]$ for $i = 1, \ldots, n$. Each core i is associated with a power table T_i that specifies the legal voltage levels for the core and the corresponding average power consumption values such that the cycle time can be achieved. These tables can be obtained by simulations that try applying different supply voltages to the core. The power consumption corresponding

[1]Since many input cores have flexibilities in shape at this stage, the restricted use of slicing floorplan can also give satisfactory results.

[2]The cost function can be easily extended to consider routability by including a term on congestion estimation.

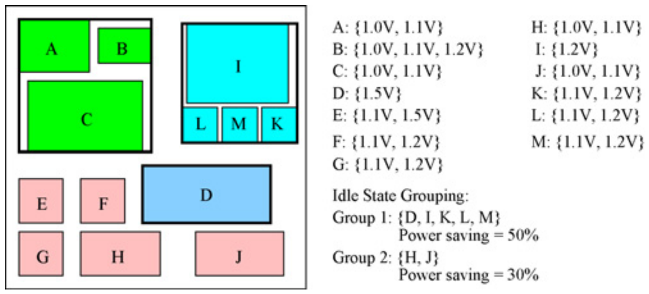


Fig. 1. Example of the voltage island driven floorplanning problem.

to each legal voltage can then be estimated.[3] We are also given a set of m nets $\{N_1, N_2, \dots, N_m\}$ and a set of groupings $\{G_1, G_2, \dots, G_p\}$ between the cores such that the cores in each group G_i have similar inactive periods and will have a s_i% saving in power consumption if they are grouped together as an island with power down mode. [4]

Given a constant K and a chip-level voltage V_c, our goal is to generate a floorplan F with K rectangular voltage islands so that the total power consumption is minimized.[5] Each island will be supplied with the lowest possible voltage level common to all the cores in that island while the remaining cores not assigned to any island will be operated at the chip-level voltage. Islands containing blocks all belonging to the same group G_i can have a further reduction in power consumption by s_i% by shutting it down during sleep.[6]

An alternative scenario of this problem is that each core i, instead of having a few discrete legal voltage levels, has a minimum voltage level $minV_i$ such that i can operate properly and achieve timing closure at a voltage level not less than $minV_i$. We call this the *minVdd* version of this voltage island-driven floorplanning problem, and will extend our discussion on this in a later part of this paper.

A. Groupings of Cores With Similar Inactive Periods

The grouping information can be generated according to the dynamic behaviors of the cores obtained from a power state machine (PSM) [8]. A PSM represents all the states a chip can operate in and the possible transitions between them. Each state in a PSM describes the operating mode of the cores. For example, a core can be active, idle or sleep in any particular state. Each state also has a weight that represents the relative amount of time the system will remain in that particular state. The larger the weight, the higher is the chance that the system will be in that state within a certain time window. The power to an island with power down mode must remain active if any of the cores in it is active. The relative amount of time a group of cores on the same island will remain inactive can be estimated from the PSM, and the percentage saving in power can then be estimated from it. We can exhaust different ways of groupings in a preprocessing step to find out which grouping is possible, given an estimate of the overhead cost in energy consumption and area of the power gating circuitry. Those potentially beneficial groups will form the G_is in the above formulation.

[3]In this paper, we assume that the power cost of a core i operated at voltage v is $v^2 A_i$, i.e., we fill in the table according to this formulation. However, the use of a power cost table gives us flexibility in specifying the power costs of a core at different voltage levels.

[4]Note that this percentage of power saving is computed according to the amount of time each core can be shut down in comparison with the total computational time, and the cores with similar inactive periods are grouped together. Therefore, even if only some cores in a group form an island, there is still s_i% saving in power.

[5]We assume that the given voltage levels of each core can meet timing, so we do not consider timing explicitly in our formulation.

[6]This factor s_i% is the percentage saving compared with the case when the cores are not shut down at all. Therefore, this is an estimation of the amount of time (compared with the total operational time) that the computations of the cores are not needed.

III. Methodology

Our floorplanner is based on simulated annealing using NPE as the representation. For each candidate floorplan solution represented by an NPE, we will perform an optimal island partitioning and voltage assignment to maximize the total power saving. The cost function of the annealing process is to minimize a weighted sum of the area, wire length, and power. We can also extend our floorplanner to consider level shifters and proximity to power pins. Details will be given in the following sections.

A. Optimal Island Partitioning and Voltage Assignment

Given a candidate floorplan solution represented by a NPE, we can construct the corresponding slicing tree and perform optimal island partitioning and voltage assignment on it. This can be done efficiently by dynamic programming. The pseudocode is shown below.

Pseudocode *TreePart(u, k)*

```
// Partition the subtree under node u into k subtrees to
// minimize the total power consumption such that the cores
// (leaf nodes) in each of these k subtree will form one island
// operated at one common voltage possibly with power down
// mode while the remaining cores not belonging to any of
// these k subtrees will be operated at chip-level voltage $V_c$

1.  $min\_cost = \infty$
2.  If k is 0, return($power(u)$).
3.  If $cost\_table[u][k]$ is updated, return($cost\_table[u][k]$).
4.  If k is 1,
5.      $C_1 = TreePart(lchild(u), 1) + power(rchild(u))$
6.      $C_2 = TreePart(rchild(u), 1) + power(lchild(u))$
7.      $C_3 = nonSubtree(u, 1)$
8.      $C_4 = cost(\{u\})$
9.      $min\_cost = \min\{C_1, C_2, C_3, C_4\}$
10.     Store $min\_cost$ into $cost\_table[u][1]$.
11.     Return ($min\_cost$).
12. Else
13.     $min\_cost = nonSubtree(u, k)$
14.     For $i = 0$ to $k$
15.         $C = TreePart(lchild(u), i) +$
            $TreePart(rchild(u), k - i)$
16.         If $min\_cost > C$, $min\_cost = C$
17.     Store $min\_cost$ into $cost\_table[u][k]$.
18.     Return($min\_cost$).
```

At the beginning, $TreePart(root, k)$ is called to obtain an optimal island partitioning and voltage assignment of the whole floorplan, where *root* is the root of the slicing tree corresponding to the NPE under consideration and k is the number of voltage islands we want to produce. When k is zero (line 2), no voltage island is formed in the subtree of u, so the power consumption $power(u)$ will be computed as $(V_c)^2 A_{\{u\}}$, where V_c is the chip-level voltage and $A_{\{u\}}$ is the total area of the cores in the subtree of u. For nonzero k, we will first check whether this optimal cost has been computed before and is available in the table for immediate return (line 3). If this value is not available, we will consider the situations when k is one and when k is larger than one separately. When k is one (line 4), there are four cases.

1) Case 1 (line 5), we continue to search for a voltage island in the left subtree of u and let the right subtree operates at the chip-level voltage V_c.
2) Case 2 (line 6), similarly, we look for a voltage island in the right subtree of u and let all the cores in the left subtree work at V_c.
3) Case 3 (line 7), use the function $nonSubtree()$ to group the cores *across* a number of subtrees along the left tree branches of u into a voltage island (details will be given in the next section).
4) Case 4 (line 8), the entire subtree u is regarded as one voltage island and the power consumption $cost(\{u\})$ will be computed as

$$cost(\{u\}) = (v_{\{u\}})^2 A_{\{u\}}$$

where $v_{\{u\}}$ is the smallest common voltage among all the cores in the subtree rooted at u and $A_{\{u\}}$ is the total area of the cores in the subtree rooted at u.

We will compute the costs of the four cases, respectively, and the smallest one will be returned and recorded in the table for future use. When k is more than one (line 12), we will recursively call the procedure $TreePart()$ to exhaust all the possible partitionings of the subtree of u, including both inter-subtree partitionings (line 13) and intra-subtree ones (lines 14–16). The minimum one will be returned and recorded in the table.

1) *Voltage Islands in nonSubtrees:* Notice that a voltage island (a rectangular region) may be formed by a set of contiguous right subtrees linked by internal nodes of the same type. An example is shown in Fig. 2. Therefore, we need the procedure $nonSubtree()$ to enumerate these cases. The pseudocode of $nonSubtree()$ is shown in the following. In this procedure, we exhaust all the cases of forming one island with two or more contiguous right subtrees and the one with the smallest power consumption will be returned. On line 7, we compute the cost of grouping the right subtrees in S as one island and having the remaining $k-1$ islands in the last left subtree (subtree D in the example of Fig. 2).

```
Pseudocode nonSubtree(u, k)
// Exhaust the cases of forming one island by a number
// of contiguous right subtrees, while the remaining k − 1
// islands are formed in the remaining left subtree.
1. min_cost = ∞
2. S = rchild(u)
3. op = operator(u)
4. While operator(lchild(u)) is op
5.    u = lchild(u)
6.    S = S ∪ rchild(u)
7.    C = TreePart(lchild(u), k − 1) + cost(S)
8.    If min_cost > C, min_cost = C
9. Return(min_cost).
```

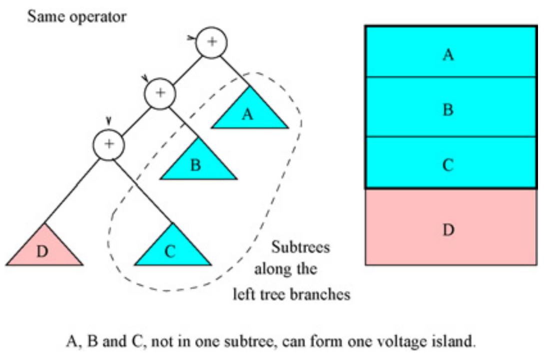

Same operator
+
+
+
A
B
C
D
Subtrees along the left tree branches
A
B
C
D
A, B and C, not in one subtree, can form one voltage island.

Fig. 2. Example of forming island across subtrees.

2) *Proof of Optimality:* The procedure $TreePart()$ will give the optimal partitioning to minimize the total power consumption. Given a candidate floorplan solution represented by a normalized slicing tree rooted at u and the number of voltage islands required k, $TreePart()$ will exhaust all the possible cases recursively and return the best solution. When k is zero, there is only one case that all the cores in the tree rooted at u (called T_u) are operated at the chip voltage. When k is one, there is only one voltage island among all the cores in T_u. The first three cases are obvious: 1) the island is in the left subtree of u; 2) the island is in the right subtree; and 3) all the cores in T_u form one island. There is still a case that the island is formed between the left and right subtrees of u. This may happen only when two consecutive internal nodes are of the same type (both "+" or both "*"). In a normalized slicing tree, an internal node will not be of the same type as its right child, so this will happen only along the left branch. $nonSubtree()$ will exhaust this last case of forming one island by a set of contiguous right subtrees along the left branch rooted at u. When k is larger than one, the cases are similar to those when k is one and $TreePart()$ will exhaust all different ways of distributing the k islands between the left and right subtrees of u and the case of having an island lying between the two subtrees. Since $TreePart()$ has exhausted all different cases of forming k islands in a given candidate floorplan, the solution returned by $TreePart()$ is optimal.

3) *Handling Island With Power Down Mode:* Islands with power down mode can be easily handled in our framework. When computing the power consumption of an island formed with the cores in the set of subtrees rooted at a node in set X by calling $cost(X)$ in the procedure $TreePart()$ and $nonSubtree()$, we only need to check if all the cores belong to one group G_i

for some $i = 1, \ldots, p$. If this is true, the island formed can be shut down during sleep and have an additional power saving of $s_i\%$.[7] Notice that an island containing just one single core can also be shut down during sleep mode if the core belongs to some idle state group. In this way, our floorplanner can give optimal island partitioning and voltage assignment taking into account islands with power down mode given any candidate floorplan solution.

4) *Speedup in Implementation and Complexity:* A table $cost\_table[v][j]$ for $j = 1, \ldots, K$ is kept at each internal node v of the slicing tree to record the optimal power consumption of partitioning the cores in the subtree rooted at v into j islands. This data structure can help to minimize the number of recursive calls and to avoid repetitive computations. It can be seen from the procedure *TreePart*( ) that whenever we want to find the optimal power saving at a node u with k voltage islands, we will first check whether this is computed before and the required information is available from $cost\_table[u][k]$. If it is available, the optimal value is returned immediately (line 3). Otherwise, it is computed recursively and the computed value will be saved in the *cost_table* to be used in some later steps (lines 10 and 17). After a move in the annealing process, only a small part of the whole slicing tree will be changed and we only need to update the tables of the affected nodes once. The affected nodes will be those lying on the paths from the modified parts of the tree to the root. For those affected nodes, the corresponding *cost_table*s will be flagged as "not updated" and will be updated during the recursive calls.

The size of the cost table is $O(nK)$ where n is the number of cores and K is the number of voltage islands, because each internal node v has K entries in the table, $cost\_table[v][j]$ for $j = 1, \ldots, K$, and there are n internal nodes in total. For each affected node v, we need to update all the K entries of its table once. Since each entry is just updated once, the time complexity will be the same as that of updating all the affected nodes in a bottom-up fashion from the leaves to the root. If the nodes were updated from the leaves to the root, the time taken to update a table at a node v was $O(K^2)$, because there were K entries and each entry took $O(K)$ time (the tables of v's children have already been updated). Therefore the total time to perform all the updates in each iteration is (number of affected nodes) $\times$ $O(K^2)$. This is $O(K^2n)$ in the worst case, and is $(K^2 \log n)$ on average.

B. Annealing Schedule

In our annealing engine, the temperature is set to 10^5 at the beginning and will drop at a rate of 0.95. At each temperature, $30*n$ random moves are performed, where n is the number of blocks in the dataset. The annealing process stops either when less than 0.5 percent random moves are accepted at a certain temperature, or when the temperature falls below 10^{-5}.

[7]Even if some cores in the group are not in the island, we assume that there is still $s_i\%$ saving in power since this percentage saving is computed based on the amount of time each core can be shut down in comparison with the total computational time. However, the way to calculate this saving in power can be modified for different problem formulations.

C. Moves

There are three kinds of moves to change the NPE of a candidate floorplan solution during annealing. This set of moves has been proven to be complete to change any arbitrary solution to any other arbitrary solution.

1) *Swap:* Swap two adjacent blocks.
2) *Complement:* Complement a chain of operators.
3) *SwapOp:* Swap a block with its adjacent operator.

D. Cost Function

We use the cost function $\psi = A + \lambda_w W + \lambda_p P$ to evaluate a floorplan where A is the area of the floorplan, W is the total wire length estimated by the half perimeter bounding box and P is the total power consumption. The parameters λ_w and λ_p are the weights which will be set at the beginning of the annealing process by random walks to make the three terms similar in weighting. Note that this cost function can be modified to consider routability (by having an additional term on congestion estimation) or the fixed-outline constraint (by replacing the area term A by $\lambda_\infty(\max\{0, w - W'\} + \max\{0, h - H'\})$ where λ_∞ is a large positive constant, w and h are the width and height of the candidate floorplan solution, and W' and H' are the fixed width and height required for the final floorplan design).

IV. Extension to Basic Methodology

Based on the framework discussed in the above Section III, the methodology can be modified to handle the following varied versions of the problem, *minVdd optimization* and *adjustable background voltage*.

A. MinVdd Optimization

As discussed in the problem formulation section, an alternative scenario of this problem is that each core i, instead of having a few discrete legal voltage levels, has a minimum voltage $minV_i$ such that i can operate properly and achieve timing closure at a voltage level not less than $minV_i$. Our general framework can be readily modified to address this *minVdd* version of the problem. The major modification made to the dynamic programming is that when we get down to the base case that the whole subtree forms only one island, instead of finding the lowest voltage level legal to all cores in the island, we should compute the highest $minV_i$ among all the cores i in the island and take that as the voltage level being used in that island. We can achieve this by computing the function $cost(\{u\})$ of a subtree u as follows:

$$cost(\{u\}) = A_{\{u\}} \times \max_{core\ i \in u} minV_i^2.$$

Notice that since we can control the number of voltage islands by adjusting the parameter k in the procedure $TreePart(root, k)$, the floorplanning process will not end up with getting more voltage levels than desired. That means, the algorithm has already taken into account the maximum number of voltage levels allowed to find an optimal way of partitioning and voltage assignment such that the total power consumption is minimized.

TABLE I
DATASETS

Data	Block No.	Net No.	Groups
$n10$	10	118	3 (30%)
$n30$	30	349	5 (30%), 5 (20%)
$n50$	50	485	5 (50%), 5 (30%), 5 (20%)
$n100$	100	885	10 (70%), 5 (50%), 6 (40%)
$n200$	200	1585	10 (50%), 10 (40%), 9 (30%), 9 (20%)
$n300$	300	1893	15 (60%), 15 (50%), 15 (40%), 15 (30%)

B. Adjustable Background Voltage

In the problem formulation, it is assumed that a background chip-level voltage V_c is given. Our framework can also be used to address an extended version of the problem in which the background voltage is not fixed yet.[8] The final chip-level voltage is determined by the minimum (or maximum) feasible voltage level among all the cores which are not grouped into any voltage island. To achieve this, we need to expand the table at each node L times where L is the total number of possible voltage levels among all the cores. In this case, each entry $cost\_table[v][j][V_l]$ will be the optimal power consumption of partitioning the cores in the subtree rooted at v into j islands when the background chip-level voltage is V_l where $1 \le l \le L$. Notice that some V_l where $1 \le l \le L$ may be impossible to be used as the background voltage in some cases and we can use a very large cost value to denote these cases. Besides the changes made to $cost\_table$, the procedures $power()$, $TreePart()$, and $nonSubtree()$ will also be modified to return an array of L values corresponding to the L cases of using a different background voltage. Steps 9–10 and steps 15–17 in the pseudocode $TreePart(u, k)$ will all become array operations treating the L different choices of background voltages separately and correspondingly.

All entries in the tables corresponding to different background voltages can be updated simultaneously during the recursive calls and the run time is thus only linearly scaled up by L. This is very affordable in practice since the number of possible voltage levels L is usually small. The memory size of such expanded cost table will be $O(nKL)$.

V. Alternatives to Solve the Problem

To further study the problem, we have developed two other methods to solve the same voltage island-driven floorplanning problem. One of them is a simplified version of the dynamic programming approach described in Section III-A, and the other one assumes a lowest possible power consumption for each module.

A. Simplified Dynamic Programming Approach

In this simplified dynamic programming approach, the islands formed across the left and right subtrees are not taken into account, i.e., step 7 and step 13 in the pseudocode of the procedure $TreePart(u, k)$ in Section III-A are removed. This approach is clearly suboptimal for a candidate floorplan,[9] since the solutions that contain islands formed across subtrees are neglected. However, the solution quality may not be significantly degraded by this simplification, since the potentially optimal grouping of some cores across subtrees in one candidate floorplan is very likely to be generated in

[8]This feature may be useful when this floorplanning tool is used as a prototype floorplanner during the front-end designing stages when the background voltage is not fixed yet.

TABLE II
COMPARISONS WITH A PREVIOUS WORK [7]

Dataset	Power Saving (%)		Dead Space(%)		Run Time* (s)	
	Ours	[7]	Ours	[7]	Ours	[7]
apte	51.64	53.78	2.341	3.422	1.457	5.482
xerox	40.45	22.85	3.667	5.259	1.335	7.079
hp	41.34	25.37	3.855	5.700	1.570	122.9
ami33	48.69	44.12	4.376	5.784	19.43	89.39
ami49	47.21	41.13	4.840	6.440	68.78	90.46
2xerox	43.68	33.53	5.136	3.765	10.63	74.75
2hp	39.58	17.47	4.880	5.650	9.55	26.86
ami75	47.95	39.06	6.552	6.330	166.9	316.5
ami99	46.33	41.16	8.630	7.666	298.5	684.4
ami200	42.68	41.90	9.110	10.88	1722	3851
ami300	41.85	40.69	10.90	12.02	3061	7380
Average	44.67	36.46	5.844	6.655	487.4	1150

*[7] is run on a Linux with a 2.1 GHz CPU and 4 GB RAM.

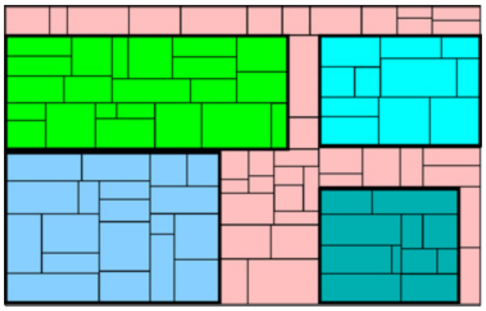


Fig. 3. One resultant floorplan of $n100$ with four voltage islands.

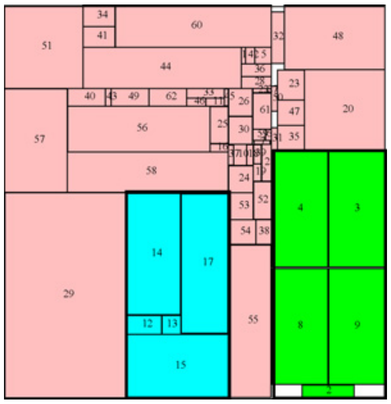


Fig. 4. One resultant floorplan of playout with some blocks operated at very low voltages.

[9]This approach is suboptimal in terms of power saving with respect to a particular candidate floorplan, but the overall simulated annealing-based optimization using this simplified approach may sometimes produce a better solution due to the randomness of the annealing process.

TABLE III
EXPERIMENTAL RESULTS WITH IDLE ISLANDS

Data	K	Total Power (x_1)	Power Saving (%)	Dead Space (%) (x_2)	Wire Length (×10) (x_3)	No. of Idle Islands	Run Time (s) (x_4)	Data	K	Total Power (x_1)	Power Saving (%)	Dead Space (%) (x_2)	Wire Length (×10) (x_3)	No. of Idle Islands	Run Time (s) (x_4)
n10	0	498 778	0.000	0.394	1185	0	3.12	n100	0	403 877	0.000	0.638	13 028	0	427.86
	1	393 660	21.075	0.296	1288	0	3.81		1	332 093	17.774	0.827	12 997	0	409.72
	2	334 853	32.865	0.748	1314	1	4.32		2	303 367	24.886	0.600	13 435	0	416.05
	3	277 862	44.291	0.748	1554	1	2.99		3	256 686	36.445	4.530	13 576	0	475.06
	4	267 263	46.416	1.561	1350	1	5.41		4	232 262	42.492	1.599	14 310	0	508.93
	5	267 263	46.416	0.487	1277	2	3.62		5	231 121	42.774	3.138	13 882	1	572.86
	6	267 263	46.416	0.407	1358	2	4.36		6	224 879	44.320	3.364	14 309	1	597.98
n30	0	469 330	0.000	0.732	3341	0	32.73	n200	0	395 316	0.000	0.808	29 421	0	1600.22
	1	312 071	33.507	0.848	3477	0	42.30		1	341 132	13.706	6.063	31 139	0	1762.17
	2	221 819	52.737	0.729	3629	1	37.43		2	309 471	21.716	1.490	30 480	0	1770.16
	3	242 060	48.424	0.760	3836	2	46.22		3	259 555	34.342	3.154	30 820	0	1814.28
	4	221 819	52.737	4.136	3803	2	43.69		4	246 987	37.522	5.704	32 004	0	2080.43
	5	221 819	52.737	4.107	4006	2	59.83		5	261 257	33.912	3.426	30 720	1	2036.82
	6	216 165	53.942	0.701	3823	2	57.28		6	247 342	37.432	0.892	32 668	1	2058.11
n50	0	446 803	0.000	0.744	8001	0	97.72	n300	0	614 632	0.000	2.527	49 839	0	3295.63
	1	267 143	40.210	3.629	8347	0	110.85		1	541 437	11.910	1.975	49 289	0	3499.20
	2	246 592	44.810	6.623	8348	0	138.71		2	494 289	19.580	2.166	50 565	0	3398.88
	3	224 870	49.671	2.141	8888	1	128.57		3	436 625	28.960	1.961	51 298	0	3576.74
	4	202 117	54.764	1.750	8296	2	140.25		4	457 999	25.484	3.514	50 383	0	3931.90
	5	227 838	49.007	0.775	8604	1	138.53		5	379 049	38.329	1.151	50 636	0	3843.81
	6	200 281	55.175	0.878	8299	3	135.08		6	386 475	37.120	2.677	52 048	1	4077.38

TABLE IV
EXPERIMENTAL RESULTS FOR SIMPLIFIED DYNAMIC PROGRAMMING APPROACH

Data	K	Total Power (y_1)	Power Saving (%)	Dead Space (%) (y_2)	Wire Length (×10) (y_3)	No. of Idle Islands	Run Time (s) (y_4)	Data	K	Total Power (y_1)	Power Saving (%)	Dead Space (%) (y_2)	Wire Length (×10) (y_3)	No. of Idle Islands	Run Time (s) (y_4)
n10	0	498 778	0.000	0.394	1185	0	3.08	n100	0	403 877	0.000	0.638	13 028	0	424.38
	1	393 660	21.075	0.345	1337	0	3.36		1	293 126	27.422	1.289	12 973	0	454.97
	2	334 853	32.865	5.801	1422	1	3.12		2	274 892	31.937	1.081	13 855	0	413.25
	3	297 459	40.362	1.320	1573	1	3.89		3	283 353	29.842	3.358	13 827	0	511.74
	4	267 263	46.416	0.759	1467	1	4.26		4	244 486	39.465	0.695	14 161	0	433.60
	5	267 263	46.416	0.875	1451	1	4.90		5	239 411	40.722	4.899	14 305	1	510.97
	6	267 263	46.416	1.566	1301	1	4.29		6	225 095	44.266	2.687	147 35	1	474.18
n30	0	469 330	0.000	0.732	3341	0	32.47	n200	0	395 316	0.000	0.808	29 421	0	1588.19
	1	312 071	33.507	0.993	3543	0	33.94		1	325 408	17.684	6.124	33 170	0	1707.47
	2	231 225	50.733	2.224	3690	1	39.28		2	324 624	17.883	3.271	31 690	0	1673.27
	3	243 983	48.015	1.159	3905	1	35.10		3	306 767	22.399	7.035	31 673	0	1667.41
	4	215 589	54.065	6.933	3892	1	42.56		4	251 292	36.433	3.809	33 361	0	1727.21
	5	209 498	55.362	0.815	3702	2	44.39		5	264 266	33.151	3.292	32 422	1	1957.23
	6	213 404	54.530	2.909	3995	3	51.93		6	261 002	33.976	5.786	34 124	1	2036.72
n50	0	446 803	0.000	0.744	8001	0	96.94	n300	0	614 632	0.000	2.527	49 839	0	3298.97
	1	265 290	40.625	0.940	8402	0	102.77		1	538 237	12.429	5.138	51 053	0	3473.72
	2	248 471	44.389	2.010	8390	0	112.04		2	487 268	20.722	1.134	49 992	0	3270.86
	3	247 515	44.603	2.544	8593	0	111.37		3	454 473	26.057	2.877	51 498	0	3373.43
	4	214 415	52.011	1.679	8024	2	130.33		4	435 452	29.152	2.246	51 824	0	3154.60
	5	206 536	53.775	3.166	8660	2	123.92		5	395 715	35.620	2.449	51 507	0	3951.88
	6	197 507	55.795	7.718	8702	3	141.29		6	407 159	33.750	4.654	53 000	1	4042.81

another one in which these cores are lying in the same subtree, due to the stochastic property of simulated annealing.

B. Lowest Possible Power Approach

In this lowest possible power approach, each core is set to operate at its lowest legal working voltage level, so that the largest possible power saving (without power down mode) can be achieved. The dynamic programming procedure in this approach then simply need to count the number of voltage islands required for each candidate floorplan, and this number will replace the power consumption term in the cost function of the annealing process. This counting can be done recursively and the pseudocode is shown below. The variables *Llevel* and *Rlevel* are used for possibly merging of two neighboring islands of the same voltage level, which are separated into two subtrees of one parent node in the slicing tree. We can easily see that the run time complexity is $O(n)$ where n is the number of cores, and this complexity is the smallest possible.

```
Pseudocode IslandCount (u, Llevel, Rlevel)
// Assume that each core i is already assigned a voltage
// level $v_i$. This procedure counts the number of voltage
// islands in the subtree rooted at u. Llevel and Rlevel
// are variables to be set and passed out of the procedure.

1. If u contains a single core i
2.   Llevel = Rlevel = $v_i$
3.   return(1)
4. Else
5.   op = operator(u)
6.   a = IslandCount(lchild(u), $L_1$, $R_1$)
7.   b = IslandCount(rchild(u), $L_2$, $R_2$)
8.   If a is 1
9.     x = $R_1$ // In this case, $L_1 = R_1$
10.    Llevel = $L_1$
11.  Else
12.    If operator(lchild(u)) equals op
13.      x = $R_1$
14.      Llevel = $L_1$
15.    Else
16.      x = 0
17.      Llevel = 0
18.  If b is 1
19.    y = $L_2$ // In this case, $L_2 = R_2$
20.    Rlevel = $R_2$
21.  Else
22.    If operator(rchild(u)) equals op
23.      y = $L_2$
24.      Rlevel = $R_2$
25.    Else
26.      y = 0
27.      Rlevel = 0
28.  If $x \neq 0$ and $y \neq 0$ and x equals y
29.    return(a + b − 1)
30.  Else
31.    return(a + b)
```

VI. Experimental Results

We have done experiments on the GSRC floorplanning benchmarks. Since no voltage information is provided in those benchmarks, we have randomly generated the voltage levels for each block from the set {1.0-V, 1.1-V, 1.2-V, 1.3-V, 1.5-V} and 1.5-V is assumed to be the chip-level voltage. In each dataset, groups of blocks with similar inactive periods are also randomly generated. Table I shows the details of each dataset, the fourth column indicates the grouping information of similar inactive periods, e.g., for $n30$, there are two groups: one contains five cores with 30% power saving if being grouped together, and the other one contains five cores with 20% power saving if being grouped together. This grouping information is now generated randomly.[10] Our algorithm is implemented in C language and all the experiments are performed on a Sun Blade 2500 with a 1.6 GHz CPU and 2 GB RAM.

TABLE V
Comparison Between Our Approach and the Simplified Dynamic Programming Approach

Data	K	Change in Total Power (%) $\left(\frac{y_1-x_1}{x_1}\right)$	Change in Total Area (%) $\left(\frac{y_2-x_2}{x_2}\right)$	Change in Wire Length (%) $\left(\frac{y_3-x_3}{x_3}\right)$	Change in Run Time (%) $\left(\frac{y_4-x_4}{x_4}\right)$
$n10$	0	0.00	0.00	0.00	−1.28
	1	0.00	0.05	3.80	−11.81
	2	0.00	5.36	8.22	−27.78
	3	7.05	0.58	1.22	30.10
	4	0.00	−0.81	8.67	−21.26
	5	0.00	0.39	13.63	35.36
	6	0.00	1.18	−4.20	−1.61
$n30$	0	0.00	0.00	0.00	−0.79
	1	0.00	0.15	1.90	−19.76
	2	4.24	1.53	1.68	4.94
	3	0.79	0.40	1.80	−24.06
	4	−2.81	3.01	2.34	−2.59
	5	−5.55	−3.32	−7.59	−25.81
	6	−1.28	2.27	4.50	−9.34
$n50$	0	0.00	0.00	0.00	−0.80
	1	−0.69	−2.72	0.66	−7.29
	2	0.76	−4.71	0.50	−19.23
	3	10.07	0.41	−3.32	−13.38
	4	6.08	−0.07	−3.28	−7.07
	5	−9.35	2.47	0.65	−10.55
	6	−1.39	7.41	4.86	4.60
$n100$	0	0.00	0.00	0.00	−0.81
	1	−11.73	0.47	−0.18	11.04
	2	−9.39	0.49	3.13	−0.67
	3	10.39	−1.21	1.85	7.72
	4	5.26	−0.91	−1.04	−14.80
	5	3.59	1.85	3.05	−10.80
	6	0.10	−0.70	2.98	−20.70
$n200$	0	0.00	0.00	0.00	−0.75
	1	−4.61	0.07	6.52	−3.10
	2	4.90	1.84	3.97	−5.47
	3	18.19	4.17	2.77	−8.10
	4	1.74	−1.97	4.24	−16.98
	5	1.15	−0.14	5.54	−3.91
	6	5.52	5.19	4.46	−1.04
$n300$	0	0.00	0.00	0.00	0.10
	1	−0.59	3.33	3.58	−0.73
	2	−1.42	−1.04	−1.13	−3.77
	3	4.09	0.94	0.39	−5.68
	4	−4.92	−1.30	2.86	−19.77
	5	4.40	1.33	1.72	2.81
	6	5.35	2.07	1.83	−0.85
Average	–	+0.95	+0.67	+1.97	−5.37

The results are shown in Table III. For each dataset, we performed voltage island driven floorplanning with the number of voltage islands generated ranges from zero to six.[11] We could see from the results that up to 50% power saving can be achieved without any significant degradation in area and wire length. In addition, the speed is very acceptable and promising. Some resultant floorplans are shown in Figs. 3 and 4. Fig. 3 is a resultant packing of dataset $n100$, with four voltage islands generated. In Fig. 4, we aim at testing a particular situation in which some cores can be operated at very low voltages. We use *playout* of the MCNC benchmark as the test case, and assign 0.6-V to cores {2–9}, and 0.8-V to cores {12–19}, respectively, as their minimum working voltage. Floorplanning

[10]This grouping information can be calculated from a task graph that shows when and how long certain cores can be powered down.

[11]Although the number of voltage levels is five, the number of islands is not limited by five because two islands can be operated at the same voltage.

TABLE VI
EXPERIMENTAL RESULTS FOR LOWEST POSSIBLE POWER APPROACH

Data	K	Total Power	Power Saving (%)	Total Area	Dead Space (%)	Wire Length ($\times 10$)	No. of Idle Islands	Run Time (s)
$n10$	4	267 263	46.416	225 032	1.490	1535	1	3.30
$n30$	8	213 796	54.446	221 016	5.622	4012	2	39.06
$n50$	7	191 360	57.171	221 552	10.369	8745	4	105.75
$n100$	20	203 397	49.639	185 450	3.208	15 778	0	326.30
$n200$	29	199 201	49.609	178 723	1.693	32 647	7	1416.87
$n300$	30	289 226	52.943	298 419	8.461	60 022	2	2723.68

is then performed with $K = 2$ and 1.5-V being the chip-level voltage. In the result, two islands are generated as expected, one with cores $\{2, 3, 4, 8, 9\}$ and the other with $\{12, 13, 14, 15, 17\}$. The cores $\{5, 6, 7\}$ and $\{16, 18, 19\}$ are not included in the islands due to other factors like interconnect and area (their sizes are small and will not cause large power wastage even if operated at a higher voltage).

A. Comparison With Previous Work

In order to compare with the results of the previous related work [7] on exactly the same problem,[12] we have done another set of experiments with the benchmarks provided by the authors of [7] without idle islands. In their datasets, the available voltage levels for each cell are chosen from the set $\{1.1-V, 1.3-V, 1.5-V, 1.8-V\}$, the area of each level shifter is 10×10 and the power cost is one. The comparisons are displayed in Table II. Results show that our approach is much more efficient and is able to save more power in most cases, while with less area overhead. Note that we set $K = 4$ in the experiments, i.e., four islands are generated for each test case.

B. Comparison With the Simplified Dynamic Programming Approach

In this simplified dynamic programming approach, the islands formed across the left and right subtrees are not taken into account, This approach is suboptimal in terms of power saving with respect to a particular candidate floorplan, but due to the stochastic property of the annealing process, the solution quality will not be significantly degraded. This observation is supported by the experiments. Table IV displays the experimental results of this simplified dynamic programming approach, while Table V compares this approach with our approach, from which we can see that this simplified dynamic programming approach performs a little bit inferior to our method in terms of solution quality, with a 5.37% reduction in running time on average.

C. Comparison With the Lowest Possible Power Approach

In this lowest possible power approach, each core is set to operate at its lowest legal working voltage level, so that the largest possible power saving can be achieved. This approach provides an upper bound on power reduction for us

[12]In [7], it is also assumed that the legal voltage levels of a core are the feasible operating voltages at which the core will satisfy its performance requirement.

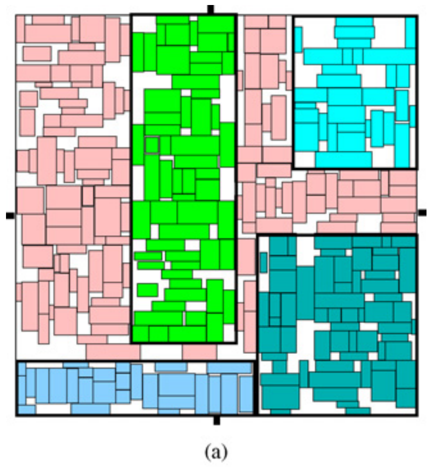
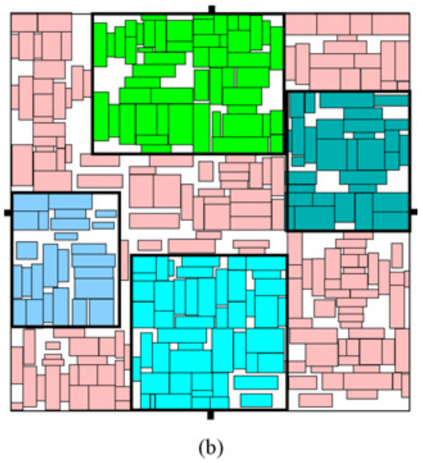



Fig. 5. Resultant floorplans for proximity constraints to power pins. (a) Without considering proximity to power pins. (b) Considering proximity to power pins.

to compare with. From Table III, we can see that the power consumption of our approach is very close to this lowest possible value, but the number of voltage islands is much smaller. Note that the efficient implementation of the island counting procedure [pseudocode *IslandCount*( )] contributes to the relatively shorter execution time of this lowest possible power approach.

D. Extension to Consider Area and Power Overhead Due to Level Shifters

Level shifters (LS) are needed for connections between two blocks in different power domains. These level shifters will

TABLE VII
CONSIDERING THE POWER AND AREA OVERHEAD OF LEVEL SHIFTERS

Data	No. of LS		Power Saving (%)		Area (Dead Space)		Run Time (s)	
	w/o LS Opt.	With LS Opt.	w/o LS Opt.	With LS Opt.	w/o LS Opt.	With LS Opt.	w/o LS Opt.	With LS Opt.
$n10$	34	15	12.42	17.83	300 394 (26.20)	256 836 (13.68)	0.667	1.095
$n30$	38	38	40.07	40.41	236 410 (11.76)	236 222 (11.70)	8.076	17.28
$n50$	103	73	28.95	36.58	242 586 (18.14)	228 655 (13.15)	23.94	63.71
$n100$	207	167	19.45	24.39	219 795 (18.33)	212 630 (15.56)	88.05	303.4
$n200$	372	316	15.93	16.94	218 822 (19.71)	209 078 (15.96)	378.2	1641
$n300$	665	460	10.95	11.53	346 578 (21.18)	308 149 (11.35)	927.6	3321
Average	236.5	178.1	21.29	24.61	260 764 (19.22)	241 928 (13.56)	237.7	891.2
Diff(%)	−24.7		+15.61		−7.22		+275	

TABLE VIII
EXPERIMENTAL RESULTS FOR MINVDD APPROACH

Data	K	Total Power	Power Saving (%)	Dead Space (%)	Wire Length (×10)	No. of Idle Islands	Run Time (s)	Data	K	Total Power	Power Saving (%)	Dead Space (%)	Wire Length (×10)	No. of Idle Islands	Run Time (s)
$n10$	0	498 778	0.000	1.158	1274	0	0.491	$n100$	0	403 877	0.000	1.052	12 536	0	85.18
	1	334 549	32.92	4.039	1293	0	0.532		1	286 710	29.01	6.767	13 294	0	85.85
	2	290 593	41.74	0.548	1463	1	0.784		2	264 563	34.49	8.728	13 222	0	88.73
	3	277 862	44.29	0.733	1438	1	0.683		3	248 192	38.55	3.791	14 205	0	90.61
	4	272 102	45.45	0.638	1340	1	0.824		4	244 842	39.38	1.918	13 957	0	95.08
	5	267 263	46.41	0.426	1359	2	0.956		5	237 291	41.24	6.848	14 138	0	99.03
	6	267 263	46.41	0.460	1320	2	0.973		6	227 722	43.62	0.891	13 725	0	103.7
$n30$	0	469 330	0.000	1.476	3420	0	7.014	$n200$	0	395 316	0.000	6.219	31 893	0	364.9
	1	275 553	41.28	0.438	3707	0	7.169		1	329 242	16.71	2.398	30 558	0	383.6
	2	271 589	42.13	0.874	3496	0	8.307		2	291 271	26.32	3.753	30 031	0	394.4
	3	221 819	52.74	3.082	3592	1	8.539		3	286 235	27.59	4.943	30 729	0	405.2
	4	221 819	52.74	0.720	3457	2	8.919		4	269 032	31.94	0.892	30 412	0	428.3
	5	212 608	54.70	0.698	3521	2	9.632		5	259 551	34.34	4.124	31 387	0	460.8
	6	209 293	55.41	0.787	3754	3	10.24		6	254 794	35.54	1.017	30 706	0	487.1
$n50$	0	446 803	0.000	0.606	7765	0	20.11	$n300$	0	614 632	0.000	2.594	51 507	0	742.9
	1	285 954	36.00	0.715	8004	0	21.19		1	393 365	36.00	4.814	49 823	0	793.8
	2	248 255	44.43	0.807	7866	0	22.15		2	379 667	38.23	1.845	47 248	0	802.3
	3	231 630	48.16	2.058	7826	0	23.46		3	361 630	41.16	2.879	53 672	0	812.7
	4	204 658	54.19	0.775	8097	1	24.66		4	353 570	42.47	1.175	48 355	0	855.6
	5	191 360	57.17	1.124	8003	2	25.41		5	350 212	43.02	6.196	50 268	0	866.1
	6	191 360	57.17	0.764	8815	3	26.87		6	342 367	44.30	3.432	50 773	0	912.4

lead to area and power overhead and should be minimized. We can extend our floorplanner to minimize the usage of level shifters by having additional terms in the cost function that represents the total area and power consumption due to level shifters. We assume that a level shifter is needed whenever a wire is connecting two blocks operating at different voltages. For example, if a net connects a source in voltage island A to three sinks, two in island B and one in island C, two level shifters will be inserted, one between A and B and one between A and C. These level shifters will add to the total area and power consumption of the design. This extra area and power usage can be estimated according to the size of a level shifter which can then be included into the area and power term of the cost function, $\psi_{\text{LS}} = (A+A_{\text{LS}})+\lambda_w W+\lambda_p(P+P_{\text{LS}})$, where A_{LS} and P_{LS} are the total area and power consumption due to the level shifters, respectively.[13] The result is shown in Table VII. In these experiments, the number of voltage islands K is set to four. Results show that our method can reduce the number of level shifters by 67.1% on average and can reduce total power consumption as well as total area, with some penalty in run time.

[13]The extra area due to the level shifters are considered and reflected in the cost function as described above. In terms of the final layout, we can proportionally increase the area of each core according to the number of level shifters inserted due to signals coming out from it. In this way, the final floorplan obtained from the annealing process can be slightly changed to take into account this extra area due to the level shifters.

E. Extension to Consider Power Network Routing

The voltage islands should be placed in proximity to the power pins to simplify the power routing step and to minimize the IR drop. The power network resources can be modeled by the sum of the half perimeters of the islands [5]. We can also extend our floorplanner to consider these power network issues by having additional terms in the cost function (with weights determined by random walk) to represent: 1) the total distance of the voltage islands from their respective power pins; and 2) the sum of the half perimeters of the islands. In our experiments, we assume that the positions of the K power pins are given. In each iteration of the annealing process, each island is matched to a power pin such that the total distance between them is the smallest possible. This total distance and the sum of the islands' half perimeters will be minimized during the annealing process.

Two resultant packings of the $n300$ dataset consisting of 300 blocks are shown in Fig. 5, and they are produced with the fixed-outline constraint. The packing in Fig. 5(a) is obtained by the original floorplaner, without taking into consideration these power network issues, while the one in Fig. 5(b) is obtained by this extended version. There are four power pins located at the center of each boundary in this example. We can see from the figures that the four islands are shifted to the sides of the chip containing the pins in order to be located closer to their respective power pins, and the islands are closer to square in shape that will favor IR drop reduction and power network routing.

F. Experiments With minVdd

In this version of the problem, instead of having a few discrete legal voltage levels, each core has a minimum voltage $minV_i$ such that i can achieve timing closure at a voltage level not less than $minV_i$. Our framework can be easily modified to address this $minVdd$ version of the problem. The results of applying our dynamic programming approach to this problem is shown in Table VIII. We can see that the running time is much shorter in comparison with Table III because it is easier to compute the maximum $minV_i$ among a group of cores than finding the smallest voltage level feasible to all of them.

VII. Conclusion

In this paper, we have proposed a simulated annealing-based approach for the floorplanning problem with simultaneous island partitioning and voltage assignment. The three factors area, wire length and power consumption of the resultant floorplan are concurrently taken into consideration, and the experiment results have shown that we are able to achieve a significant power saving of up to 50% for the testing datasets.

In addition, when extended to minimize the number of level shifters between different voltage domains, our method can reduce 67.1% of the LS used on average, with some penalty in power saving; and it also functions well to generate voltage islands in proximity to their corresponding power pins, by having an additional term in the cost function to minimize the total distance between voltage islands and power pins.

References

[1] Y. Cai, B. Liu, Q. Zhou, and X. Hong, "Voltage island generation in cell-based dual-vdd design," *Inst. Electron., Informat. Commun. Eng. Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E90-A, no. 1, pp. 267–273, 2007.

[2] L. Guo, Y. Cai, Q. Zhou, and X. Hong, "Logic and layout aware voltage island generation for low power design," in *Proc. Asian South Pacific Design Automat. Conf.*, 2007, pp. 666–671.

[3] J. Hu, Y. Shin, N. Dhanwada, and R. Marculescu, "Architecting voltage islands in core-based system-on-a-chip designs," in *Proc. Int. Symp. Low Power Electron. Design*, 2004, pp. 180–185.

[4] W.-L. Hung, G. M. Link, Y. Xie, N. Vijaykrishnan, N. Dhanwada, and J. Conner, "Temperature-aware voltage islands architecting in system-on-chip design," in *Proc. Comput. Design*, 2004, pp. 689–696.

[5] W.-P. Lee, H.-Y. Liu, and Y.-W. Chang, "Voltage island aware floorplanning for power and timing optimization," in *Proc. Int. Conf. Comput.-Aided Design*, 2006, pp. 389–394.

[6] Q. Ma and E. F. Y. Young, "Voltage island-driven floorplanning," in *Proc. Int. Conf. Comput.-Aided Design*, 2007, pp. 644–649.

[7] W.-K. Mak and J. W. Chen, "Voltage island generation under performance requirement for SoC designs," in *Proc. Asian South Pacific Design Automat. Conf.*, 2007, pp. 798–803.

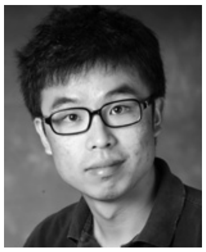
[8] D. Sengupta and R. Saleh, "Application-driven floorplan-aware voltage island design," in *Proc. 45th Assoc. Comput. Machinery/IEEE Design Automat. Conf.*, 2008, pp. 155–160.

[9] X. Tang, R. Tian, and D. F. Wong, "Minimizing wire length in floorplanning," *IEEE Trans. Comput.-Aided Design Integrat. Circuits Syst.*, vol. 25, no. 9, pp. 1744–1753, Sep. 2006.

[10] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. 23rd Assoc. Comput. Machinery/IEEE Design Automat. Conf.*, 1986, pp. 101–107.

[11] H. Wu, I.-M. Liu, D. F. Wong, and Y. Wang. "Postplacement voltage island generation under performance requirement," in *Proc. Int. Conf. Comput.-Aided Design*, 2005, pp. 309–316.

[12] S. Yang, W. Wolf, N. Vijaykrishnan, and Y. Xie. "Reliability-aware SoC voltage islands partition and floorplan," in *Proc. Emerging Very Large Scale Integrat. Technol. Architect.*, 2006, p. 343.



Qiang Ma received the B.Eng. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2006, and the M.Phil. degree in computer science from the Chinese University of Hong Kong, Shatin, Hong Kong, China, in 2008. He is currently working toward the Ph.D. degree at the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign.

His research interests include physical design of chips, packages, and printed circuit boards.

Evangeline F. Y. Young received the B.S. and M.Phil. degrees in computer science from the Chinese University of Hong Kong (CUHK), Hong Kong, China, and the Ph.D. degree in computer science from the University of Texas, Austin, in 1999.

From 1999 to 2004, she was as an Assistant Professor with the Department of Computer Science and Engineering, CUHK, and currently she is an Associate Professor in the same department. Her current research interests include algorithms and computer aided design of very large scale integration circuits. She is now working actively on floorplanning, placement, routing, and algorithmic designs.

Dr. Young has served on the technical program committees of several major conferences including the International Conference on Computer Aided Design, the Asia and South Pacific Design Automation Conference, the International Symposium on Physical Design, and the Great Lakes Symposium Very Large Scale Integration.